INFORM
Reference Guide
PDR 3905

First Edition

by
Laura J. Douros

This book documents the operation of Release 5.0 of INFORMATION software which runs on all Prime Information Systems. Release 5.0 is based on PRIMOS Master Disk Revision Level 18.2 (Rev. 18.2).

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

COPYRIGHT INFORMATION

HOW TO ORDER TECHNICAL DOCUMENTS

U.S. Customers

Software Distribution
Prime Computer, Inc.
1 New York Ave.
Framingham, MA 01701
(617) 879-2960 X2053, 2054

Customers Outside U.S.

Contact your local Prime subsidiary or distributor.

Prime Employees

Communications Services
MS 15-13, Prime Park
Natick, MA 01760
(617) 655-8000, X4837

INFORMATION Systems

Contact your Prime INFORMATION system dealer.

PRINTING HISTORY — INFORM Reference Guide

Edition	Date	Number	Documents
Pre-release	March 1979	IDR3905	Release 1.0
Preliminary	March 1980	IDR3905 Revision 2	Release 3.0
First Edition	November 1981	PDR3905	Release 5.0

SUGGESTION BOX

All correspondence on suggested changes to this document should be directed to:

Laura J. Douros
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

Contents

About This Book

This book covers all of INFORM's verbs, keywords, and options. As some familiarity with INFORMATION terms and concepts is assumed, users new to the system should read the INFORMATION Highlights Companion before using this book.

Section 1 establishes the framework in which INFORM operates. It explains what INFORM is and does and how it fits in with other parts of the INFORMATION system.

Section 2 covers all the important INFORMATION file concepts. Written to accommodate the needs of both new and experienced users, it explains what the various parts of an INFORMATION file are and what they do. Many examples are included to show you how to create, modify, and work with INFORMATION files.

Section 3 discusses all the INFORM verbs, what they do, and what options and keywords can be used with them.

Section 4 documents all the INFORM keywords. It divides and lists them both alphabetically and by function. Although this section contains reference material, its style is tutorial in that each keyword is accompanied by an example showing how it can be used.

ACKNOWLEDGEMENTS

The author would like to thank the members of the Dealer Program, including the dealers and their customers, and the members of Technical Publications, who patiently reviewed this book and contributed their suggestions to it.

INFORMATION DOCUMENTATION CONVENTIONS

The following conventions are used in command or verb formats and in examples throughout this book and in other INFORMATION documentation. Command or verb formats show the general usage and syntax of INFORM verbs. Examples show how these verbs can be combined with the various keywords and options in typical applications. Terminal input may be entered in either uppercase or lowercase.

Convention	Explanation	Example
UPPERCASE	In command formats, words in uppercase indicate the actual names of commands, statements, and keywords. They can be entered in either uppercase or lowercase.	LIST filename
abbreviations	If a command or statement has an abbreviation, it is indicated by underlining.	TOTAL
lowercase	In command formats, words in lowercase indicate items for which the user must substitute a suitable value.	SORT filename
underlining in examples	In examples, user input is underlined and system prompts and output are not.	:LIST CUSTOMER

Brackets [ ]	Brackets enclose one or more optional items. Choose none or one of these items.	LIST [DICT] filename
Braces { }	Braces enclose a vertical list of items. Choose one and only one of these items.	{LIST|filename {SORT}
Ellipsis	Indicates that the preceding item may be repeated one or more times.	[BY.DSND field]...
Parentheses ( )	In descriptor expressions parentheses should be entered exactly as shown.	TOTAL(COST/QTY)
(CR)	Indicates a single carriage return. This is accomplished by hitting the RETURN key or NEWLINE key, depending on the type of terminal you have.	

SECTION 1

INTRODUCTION TO INFORM

INTRODUCTION

This section is an introduction to the INFORM processor on Prime INFORMATION Systems. INFORM is only a piece of the total INFORMATION package, and cannot be fully appreciated without looking at it in relation to the rest of the system. A brief overview of the entire system can be found in the INFORMATION Highlights Companion. It is highly recommended that you read the companion before attempting to use this book.

About This Section

Section 1 introduces INFORM, tells you what it can do, and explains how to communicate with it. The following topics are covered:

- The INFORM Processor -- what it is and what it does
- Communicating with INFORM -- rules for INFORM sentences
- The PERFORM Sentence Stack -- what it is and how to use it

THE INFORM PROCESSOR

INFORM is the information manager, query language, and report generator for the INFORMATION system. It takes care of storing, retrieving, and formatting data in a way that nonprogrammers can easily understand.

INFORM is an Easy-to-Use Tool

Although there are other methods of accessing stored data in INFORMATION, such as INFO/BASIC programs, INFORM is by far the most complete and easy-to-use data access tool available to the INFORMATION user. You don't have to know anything about data storage, data access methods, or programming to use INFORM. All you have to know are a few verbs (command words), and keywords (option indicators), and the names of the files from which you want to retrieve data. Some knowledge of INFORMATION terms is assumed. Therefore, it is recommended that you first read the INFORMATION Highlights Companion to familiarize yourself with the important INFORMATION terms and concepts used throughout this book.

How INFORM Fits in With PERFORM

Every command, or directive, that you give to INFORM is first received and interpreted by PERFORM, which serves as INFORMATION command environment monitor. It is a monitor in the sense that it waits for you to give it some input and then decides what part of the system should handle the request. Once you have entered the INFORMATION system, a process which is fully described in the INFORMATION Highlights Companion, you are at PERFORM command level. From this point on, PERFORM picks up everything you type in response to the colon prompt (:) and decides what should be done with it. Thus, all directions given to INFORM are first passed through PERFORM.

What INFORM Can Do

Although there are only five major verbs in INFORM that you are likely to use often, they can be combined with one or more of about 58 keywords and options that are part of INFORM. Such combinations of verbs and keywords can do any kind of search, display, calculation, and output formatting imaginable. Here are just some of the things that you can do in INFORM:

- Define the logical structure of data stored in an INFORMATION file.
- Add, delete, and modify data in an INFORMATION file.
- Search a file for one or more records that match one or more specified conditions.
- Search for a record with a data value that "sounds like" some phonetic spelling.
- Sort a list of records by some field in ascending or descending order.
- Create a report with headers, footers, page numbers, calculation breakpoints, etc.
- Display or print file information with double-spacing, column suppression, margin specification, and vertical alignment.
- Calculate averages, percents, and totals for numeric data.
- Put breakpoints in reports to display partial sums and averages.

- Create a list of selected records and perform one or more operations on all the records in this list.
- Use INFO/BASIC subroutines that you've written in calculating new field values.
- Pull values from more than one file for use in evaluating conditions or in evaluating a particular field value.

Of course, these are just a few of the things you can do with INFORM. The possibilities are virtually unlimited because of the number of keywords available and the number of ways in which they can be combined.

COMMUNICATING WITH INFORM

Directions to INFORM are very much like standard English imperatives. Some examples are: "Wash your car," "Mow the lawn," and "Send money." Except for words like "the" and "your," INFORM sentences follow the same format that these imperative sentences do. They are simply composed of an action verb and an object to which the action is being applied.

Every INFORM sentence consists of at least a verb, also called a command, and a filename. The verb specifies the action and the filename specifies the object on which the action should be performed. In addition, sentences can include field names and options to specify more completely what fields in a record an operation should be applied to and what kind of output or display options should be used.

Sentence Format

Sentence structure in INFORMATION is flexible, as the examples used in this book indicate. There are a few suggestions for sentence construction that make it easier for other users to interpret sentences that you've constructed. The general format of a sentence can be represented as shown in the text below and in Figure 1-1. Remember, this is a suggested format and is intended to help you understand how INFORM interprets sentences.

verb filename [field names] [selection criteria] [sort options] [options] [record IDs]

Only a verb and a filename are required. The other items are optional, which is why they are shown in brackets.

REQUIRED OPTIONAL *

VERB FILENAME	[DISPLAY FIELDS]	[SELECTION CRITERIA]	[SORT OPTIONS]	[OUTPUT OPTIONS]	[REPORT OPTIONS]	["RECORD IDs"]

VERB: Specifies one of these INFORM verbs:

COUNT
LIST
SELECT
SSELECT
SUM

FILENAME: Specifies a data or Dictionary file:

FILENAME
or
DICT FILENAME

* Options can be specified in any order.

DISPLAY FIELDS: Specifies fields to be displayed, or name of phrase specifying display fields.

SELECTION CRITERIA: Specifies search conditions to be met using one of these phrases:

{WHEN / WITH} condition
{WHEN / WITH} [EVERY] condition

SORT OPTIONS: Specifies sort fields using these keywords:

{BY / BY.DSND / BY.EXP / BY.EXP.DSND} sort.field

OUTPUT OPTIONS: Specifies suppression of INFORM display defaults using keywords like:

COL.SUP
ID.SUP
NO.PAGE
(etc.)

REPORT OPTIONS: Specifies report and formatting options using keywords like:

BREAK.ON
CALC
FOOTING
HEADING
(etc.)

"RECORD IDs": Specifies record ID values; quotes recommended for clarity

Figure 1-1. General INFORM Sentence Format

field names are the names of fields in the file that you want displayed. The selection criteria argument specifies the condition or conditions that record values must meet in order to be chosen. sort options specify the field or fields by which the output will be sorted. The options argument defines any other output or display options. All of these options are discussed in Sections 3 and 4. Because of INFORM's flexible sentence format, any of the items just defined can appear anywhere in a sentence after the verb and filename have been specified.

record IDs are actual record ID values in the file. Like field names and sort options, record IDs can appear anywhere in the format. Putting them at the end of the sentence and enclosing them in quotes, as in:

```
LIST CUSTOMERS LNAME FNAME "DARCY" "MOLLICA"
```

makes it a bit easier to read and helps distinguish between the field names and record ID values. This sentence indicates that the records which have key values of DARCY and MOLLICA should be found and displayed. Because the field names LNAME and FNAME were specified, only the values in LNAME and FNAME will be displayed.

Rules for INFORM Sentences

Here are the general rules that govern the format of INFORM sentences:

1. Sentences should be entered in response to the PERFORM colon prompt (:).

2. INFORM sentences are executed by hitting a carriage return. A carriage return is accomplished by hitting the RETURN or NEWLINE key on the keyboard, depending on your type of terminal.

3. INFORM sentences must begin with a verb defined in your VOC file. Every user account has a VOC file which defines all the verbs, keywords, and filenames that can be used or accessed from that account. Whenever a new account is created, a copy of the master VOC file, called NEWACC (in the ISYS file), is put into the new account. The user of the new account can then define new sentences to put into this version of the VOC file. Synonyms for existing keywords or verbs that are already defined in the master copy of the VOC file can also be defined and put in the user's version of the VOC file. For additional information on the VOC file, see the INFORMATION Highlights Companion.

4. INFORM allows only one filename per sentence. The file must be defined by a file definition entry in the user's VOC file.

5. INFORM sentences may contain any number of record IDs, field names (including both D-descriptors and I-descriptors), phrases, and keywords. Field names must be defined in the dictionary of the file that is being referenced. Phrases must be defined either in the dictionary of the file being referenced, or in the VOC file. Keywords must be defined in the VOC file.

6. Sentences that are longer than the width of a terminal screen (usually 80 characters) can be continued onto several lines by appending a back arrow or underscore character (_) to the end of each line, and following it by a carriage return. PERFORM will prompt for each additional line of the sentence with the right angle (>) prompt. The last line of the sentence should not be terminated by a back arrow. Instead, simply hit a carriage return to signal the end of the sentence.

INFORM Sentence Evaluation

INFORM evaluates all user input in the same way. It expects sentences to have some sort of format that can be analyzed. The analysis process is outlined in these three steps:

Step 1: The first item in an INFORM sentence is always interpreted by INFORM as a verb. This item is looked up in the VOC file, and if a match is found, INFORM checks to see that it is indeed a verb. The verb can be a system-defined verb or a user-defined verb that has been added to the VOC file. If the item is not a verb, as in this example, a message similar to this one is displayed:

```
:BY.DSND BIRTHDAYS
BY.DSND in VOC file has illegal type of K.
```

If the item is not found at all, as in this example, a message similar to this one appears:

```
:SEND MONEY
SEND is not in your vocabulary file.
```

Step 2: The second item in a sentence is assumed to be a filename. Again, INFORM looks up this item in the VOC file. If this item is not found in the VOC file, a message like this one is displayed:

```
Sorry, but your sentence does not contain a recognizable file name.
```

If this item is found in the VOC file, it must have a type code of F. If it doesn't, it is not a legal file entry, and the same message will be displayed.

Step 3: Having verified that the sentence so far contains a legal verb and filename, INFORM uses the dictionary associated with the file to process the rest of the sentence. This means that everything else in the sentence besides the first two items is assumed to be defined in the dictionary. Thus, INFORM looks up all remaining pieces of the sentence in the dictionary. If an item isn't located here, it is looked up in the VOC file. When an item is not found in either file, INFORM interprets it as a record ID value from the data file. INFORM then tries to do a search of the data file using this record ID. If the search fails, a message like this appears:

```
:LIST BIRTHDAYS LNAME FUDGE

Press <NEW LINE> to continue...
LIST BIRTHDAYS LNAME FUDGE 14:04:38  08-13-81  PAGE     1
RECORD ID     Last Name.....

No records found on the BIRTHDAYS File.

"FUDGE" not found.
```

General Notes

In addition to the rules noted earlier, you will want to be aware of the following suggestions regarding the construction and use of INFORM sentences.

- Record ID values in a sentence can be surrounded by pairs of double quotes (") or single quotes ('apostrophes') to make the sentence easier to read. This helps distinguish field values stored in the data file from field names and keywords which are defined in the dictionary or VOC file.

- Field names are generally processed by INFORM in the order in which they appear in the sentence.

- When listing a record in the VOC file, be sure to enclose its name in matching quotes or apostrophes. For example, to list the contents of the COSTS record (which is a file definition entry in the VOC file), do not type:

  LIST VOC COSTS

  Instead, type:

  LIST VOC "COSTS"

- Quotes or apostrophes must be used around values that contain embedded spaces. For example, this form of a sentence:

  LIST CUSTOMERS WITH NAME EQ TOM SMITH

  would cause the following error messages to be generated:

  "TOM not found on the CUSTOMERS file"

  "SMITH not found on the CUSTOMERS file"

  The field value TOM SMITH contains an embedded space between TOM and SMITH. The proper format of this sentence would be:

  LIST CUSTOMERS WITH NAME EQ "TOM SMITH"

- Keywords provide the user with control over the default actions of the specified verb. Although keywords often have a required syntax, the entire keyword clause can usually appear anywhere in the sentence.

Sample INFORM Sentences

Here are a few sample INFORM sentences.

The first sentence lists all the records in the PERSONNEL file that meet the specified conditions. ANNUAL.GROSS and DEPENDENTS are field names in the PERSONNEL file.

```
LIST PERSONNEL WITH ANNUAL.GROSS < 10000 AND DEPENDENTS > 2
```

This sentence lists the contents of a record with an ID value of "COMMISSION" in the SALES file dictionary:

```
LIST DICT SALES "COMMISSION"
```

To create a list of all the records in the DEPARTMENTS file with names containing the word PAYROLL, you could use this sentence:

```
SELECT DEPARTMENTS WITH NAME MATCHING ...PAYROLL...
```

The following sentence counts the number of records in the CUSTOMERS file that have a credit balance of greater than $1000:

```
COUNT CUSTOMERS WITH CREDIT.BAL > 1000
```

The SUM verb adds up the values in a given field in a file. In this sentence, the values in the SHIPPING.WEIGHT and SHIPPING.COST fields of the ORDER file are summed:

```
SUM ORDERS SHIPPING.WEIGHT SHIPPING.COST
```

Sums for both of the fields are then displayed.

These are only a few of the things you can do with INFORM. Sections 3 and 4 contain additional examples of INFORM sentences.

THE PERFORM SENTENCE STACK

Because INFORM sentences are typed in response to the PERFORM colon prompt, you can take advantage of a handy feature of the PERFORM operating environment, called the PERFORM sentence stack. The stack, as it is sometimes referred to, keeps a record of all the commands you enter at PERFORM command level. Every command and INFORM sentence that you type from the terminal in response to the PERFORM colon prompt (:) is saved in this stack, which can contain up to 99 entries. The oldest entries are constantly being overwritten by new ones. Thus, at any given time, the most recent 99 commands you've entered are stored in your stack.

Stack Entry Numbers

Each line, or entry, in the stack is numbered with a 2-digit number from 01 to 99. The "oldest" entry in the stack is entry number 99, and the newest, or current, entry is entry number 01. Stack entries are also referred to as "stack sentences," "stack lines," or "stack items." This book calls them stack entries for consistency.

Stack Commands

There is a special set of commands for manipulating the stack. They are briefly summarized in the INFORMATION Highlights Companion, and are discussed in greater detail in the PERFORM Reference Guide. They are called "dot" commands, because each one must be preceded by a dot (.).

Here is a brief explanation of what you can do with these commands.

Looking at the Stack: To take a look at the most recent commands you've typed in response to the colon prompt, use this command:

```
.L [nn]
```

Without the nn argument, this command will give you a list of the last 20 commands and sentences you've executed. To see more than 20 lines, supply the appropriate number for nn. Then nn lines will be listed instead of 20.

Recalling a Stack Entry: To recall any stack entry or a sentence or paragraph from your VOC file to be modified or re-executed, type:

```
.R [nn
    sentence.name
    paragraph.name]
```

The argument nn stands for the number of the stack entry you want recalled. sentence.name is the name of a sentence in your VOC file, and paragraph.name is the name of a paragraph in your VOC file. A recalled sentence from your stack or a single sentence from your VOC file becomes entry number 01 in the stack, bumping all the other stack entries up by 1. When a paragraph is recalled, the sentences in this paragraph are placed in stack positions 01 through nn, where nn is the number of executable sentences in the paragraph. Once you've recalled sentences to the stack, you can edit them and then re-execute them.

Modifying Stack Entries: To modify any stack entry, use the .C command. You can use practically any special character as a delimiter, as long as you are consistent within a single change command. In this format, the / (slash) character acts as a delimiter:

.C[nn]/str.old/str.new/ [g]

If nn is specified, the change is made in stack entry number nn, otherwise the change is made in stack entry 01. str.old is an existing string in the sentence, and str.new is what you want the old string to become. The g option changes all the occurrences of str.old to str.new in the current stack entry.

Re-executing Stack Entries: Use the .X[nn] command to execute the stack entry of your choice. If nn is not supplied, the current stack entry, number 01, will be executed.

Storing Stack Items: The .S command allows you to save INFORM sentences in your VOC file for future use. This saves time and typing because all you have to do to execute a stored sentence is type its name. Even better is the ability to store a series of executable (legal) INFORM sentences in a paragraph under a single name.

The syntax of the .S command is:

.S name [[n1][n2]]

The name parameter is the name you want to give this sentence or series of sentences which will be stored in your VOC file. The n1 and n2 parameters stand for the beginning and ending line numbers of sentences in the stack. For instance, the sentence:

.S NEIL1 6 4

will save stack entries 6, 5, and 4 under the name NEIL1 in your VOC file. A paragraph entry will be created in this case because there is more than one sentence being saved.

If you don't specify any numbers for the n1 and n2 arguments, the most recent stack entry, which is entry number 01, is saved.

If you specify just n1, the n2 argument defaults to 01. For instance, if n1 is specified as 5, and n2 is not supplied, lines 5, 4, 3, 2, and 1 will be saved.

Suppose this sentence was typed:

.S LAURA2 8

Stack entries 01 through 08 would then be saved in the VOC file as a paragraph under the name LAURA2. Be careful of this! If you only want to save a single stack item, for example, stack entry 08, recall it to position 01 and then save it.

The .S command automatically causes a new record to be added to the VOC file. This record contains the text of the saved sentence or sentences. When there is only one from the stack being saved, an S appears in Field 1 of the VOC entry entry to indicate that it is a stored sentence. If you save more than one sentence, the new record entry will have a PA (for paragraph) in Field 1.

If an entry already exists in the VOC file with the same name that appears in the .S command line, a message will be displayed indicating that name is already in use.

Executing Stored Sentences: To execute a stored sentence or paragraph, simply type in the name that you specified when you saved the entry or entries with the .S command. Enter the name at the PERFORM colon prompt and the sentence, or series of sentences, will be executed. The command you typed to make this happen will be put in position 01 in the stack, but the actual contents of the sentence or paragraph entry will not be placed in the stack.

Deleting a Stack Entry: To remove an entry from the stack, use the .D command.

.D [nn | sentence.name | paragraph.name]

When the nn option is specified, only the entry in position nn on the stack will be deleted. If either sentence.name or paragraph.name is specified, the appropriate VOC sentence or paragraph entry will be deleted from the VOC file. If no options are specified, the current stack entry, which is always entry 01, will be deleted. When nn is specified, that stack entry will be removed, and all the stack entries that were stored above this entry will be bumped down by 1.

SECTION 2

DICTIONARY CONCEPTS

INTRODUCTION

This section explains the purpose and structure of an INFORMATION file. It describes in detail the two basic parts of each INFORMATION file, the dictionary and the data file. Because of the complexity and importance of dictionary files, the bulk of this section deals with dictionary concepts. Various methods of creating and modifying both dictionary files and data files are also outlined here and are accompanied by many examples to help you use them. Much of the material covered here assumes that you have already read or are familiar with the basic concepts covered in the INFORMATION Highlights Companion. However, for those who have not read the companion, this section begins with a brief review of some important INFORMATION terms that are used throughout this book.

What's in This Section

The topics covered in this section are:

- Brief Review -- some INFORMATION file concepts
- Dictionary Files -- definition and purpose
- Creating a Dictionary File -- the steps involved
- Dictionary File Contents -- what dictionaries contain
- Data Descriptors -- purpose and structure
- @ID Descriptors -- default record ID descriptors
- I-descriptors -- purpose and structure
- I-descriptor Expressions -- the components available for creating expressions
- Tools for I-descriptor Expressions -- the complete repertoire of I-descriptor expression options
- Phrases -- structure and definition
- Data Files -- purpose, structure, and data entry techniques

BRIEF REVIEW

Every INFORMATION file has two parts:

- A data file, which contains the actual information you're interested in storing, retrieving, updating and printing out.
- A dictionary file, which is a logical description of the data file. It defines the fields in a typical file record and describes how each field should be displayed.

Before creating a file, you need to decide what elements of information it's going to contain and what you are going to call them. As described in the INFORMATION companion, there are three major steps involved in creating an INFORMATION file once you decide what you want in it:

1. Defining the data/dictionary file with CREATE.FILE verb
2. Defining the contents of the dictionary file with ENTRO
3. Adding data to the file with ENTRO, INFO/BASIC programs

Both INFORMATION dictionary files and data files are described in this section. It explains how to set them up, how they interact, and what they look like.

DICTIONARY FILES

An INFORMATION dictionary file describes the data that is stored in a separate data file. Data files, like most other files you are probably familiar with, store information in units called records. Each record is composed of a number of fields, or elements of data. The values contained in these fields usually differ from one record to another.

The purpose of a dictionary is to define the structure of a data file record. This definition includes a description of all the fields in a data file record, their positions in the record, their output format, and whether or not each field can contain more than one value in a single record.

Each dictionary file has the same physical structure. It is made up of records that describe fields in the data file associated with it.

There are three kinds of dictionary file records:

- Data descriptors, which describe fields in the data file.
- I-descriptors, which describe special "calculatable" fields.
- Phrases, which are sentence fragments that do not contain verbs.

Data Descriptors

Each field in a data file is described by a data descriptor, which is also called a field name. There can only be one data descriptor, called a D-descriptor, for each field in a data file record. Data descriptors make it possible to specify which fields in a data file can be searched on, changed, or printed out. It is possible to have synonyms, or more than one descriptor associated with a given field in the data file, but you must use an I-descriptor to do this.

I-descriptors

I-descriptors describe information that does not necessarily exist in the data file. Instead, they describe how to derive information from one or more data items that are actually stored in the file. These descriptions are in the form of expressions that can be evaluated to produce some meaningful information. Such expressions can involve arithmetic or string operations on fields stored in one data file or in several data files. INFORMATION provides many predefined functions and subroutines which can be included in I-descriptor expressions.

During data entry, the user is not prompted for input to I-descriptor fields. The value for an I-descriptor is obtained by referencing its name in an INFORM sentence and looking at the output resulting from that sentence. Whenever an I-descriptor is referenced, the expression defined for this descriptor is evaluated to obtain a value for the field.

Phrases

A phrase makes it possible to refer to several data fields in a file with just one name. These data fields can be D-descriptors, I-descriptors, or even other phrases, as long as they all exist in the same file.

Like descriptors, phrases are generally defined in the file dictionary. However, a phrase also can come in handy when you want to use part of a sentence with more than one INFORM verb or file. The phrase would then be stored in the VOC file. Most phrases are file specific though, and are stored in file dictionaries. For example, the phrase:

```
WITH GRAD.YR AFTER 1981 LNAME GRAD.YR
```

could be stored in the STUDENTS file under the name FUTURE.GRADS. The GRAD.YR field contains the year in which the student is expected to graduate. If the value in the GRAD.YR field is greater than (AFTER) the value 1981, the record will be chosen and displayed. The AFTER keyword performs a "greater than" comparison, much like the > relational operator. The output will include the record ID field (default), and the LNAME and GRAD.YR field values for each record chosen.

Defining a phrase like the one just described makes it possible to transform the following sentence:

```
LIST STUDENTS WITH GRAD.YR AFTER 1981 LNAME GRAD.YR
```

into the form:

```
LIST STUDENTS FUTURE.GRADS
```

How Phrases Are Used: A phrase can also be a shorthand way of referring to more than one field or descriptor with a single name. For instance, the phrase EMP.NAME could be used to refer to the fields LNAME, FNAME, MID.INIT, and TITLE in an employee history file. Any reference to EMP.NAME in an INFORM sentence that applies to this file will automatically refer to all the fields defined in the phrase.

Phrases are generally plugged into sentences that already contain a verb and a filename. In a dictionary file, you can define phrases to reference any combination of fields that you want. Phrases defined in a dictionary file are referenced by a user-defined name, just like phrases stored in the VOC file.

Record ID Descriptor

Every record in an INFORMATION data file has a unique key called a record ID. The characteristics of this key (numeric, alphanumeric, strictly alphabetical, and so forth) determine the file type you specify when creating an INFORMATION file with the CREATE.FILE command. (See the PERFORM Reference Guide for details on this command.)

In each dictionary, there is a special data descriptor for the record ID. It is called @ID and is automatically added to the dictionary file when the file is created. The record ID descriptor is a "D" type descriptor and defines Field 0 of each data file record. Because the @ID record is automatically created when the file is allocated, its record position, display name, and output format are all established by default. It is possible to change some of the defaults, such as the output format specification or the display name. Do not change the record position, or relative location, of the @ID descriptor, as it should always define Field 0 of the data record.

Sample Dictionary File

The following example lists the contents of a dictionary file which describes a data file called STUDENTS. In Example 2-1, as in all subsequent examples, user input is underlined to distinguish it from system output. The STUDENTS file is used as an example throughout this book.

```
:LIST DICT STUDENTS

FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@ID            D    0                  SOC-SEC-NO      11L    S
LNAME          D    1                  LAST NAME       15L    S
FNAME          D    2                  FIRST NAME      15L    S
MI             D    3                  MID. INIT.      2L#.   S
GRAD.YR        D    4                  GRAD YEAR       4L     S
COURSES        D    5                  COURSES TAKEN   12T    M  COURSE.GRA
                                                                  DES
                                       BY STUDENT

6 records listed.
```

Example 2-1. Sample Dictionary File

CREATING A DICTIONARY

There are two simple steps to follow when creating the dictionary for a data file:

1. Use CREATE.FILE to set up a dictionary and a data file.
2. Define the dictionary file contents with ENTRO.

The CREATE.FILE Verb

The CREATE.FILE verb may be used in one of three ways, depending on which "parts" of an INFORMATION file you want to set up:

You Can:	By Typing:
Create both data file and dictionary file at the same time	CREATE.FILE filename
Create dictionary file only	CREATE.FILE DICT filename
Create the data file only	CREATE.FILE DATA filename

filename is the name by which the file will be referenced. The dictionary file is automatically created with the default name D_filename, where filename is the name of the file being created.

More on CREATE.FILE: All of these formats invoke the long form, or interactive version, of this command. The short form can be used if you want to supply all the necessary information on the command line, without having CREATE.FILE prompt you for it. Details on the CREATE.FILE command, its variations and its functions, can be found in the PERFORM Reference Guide.

About the Dictionary File

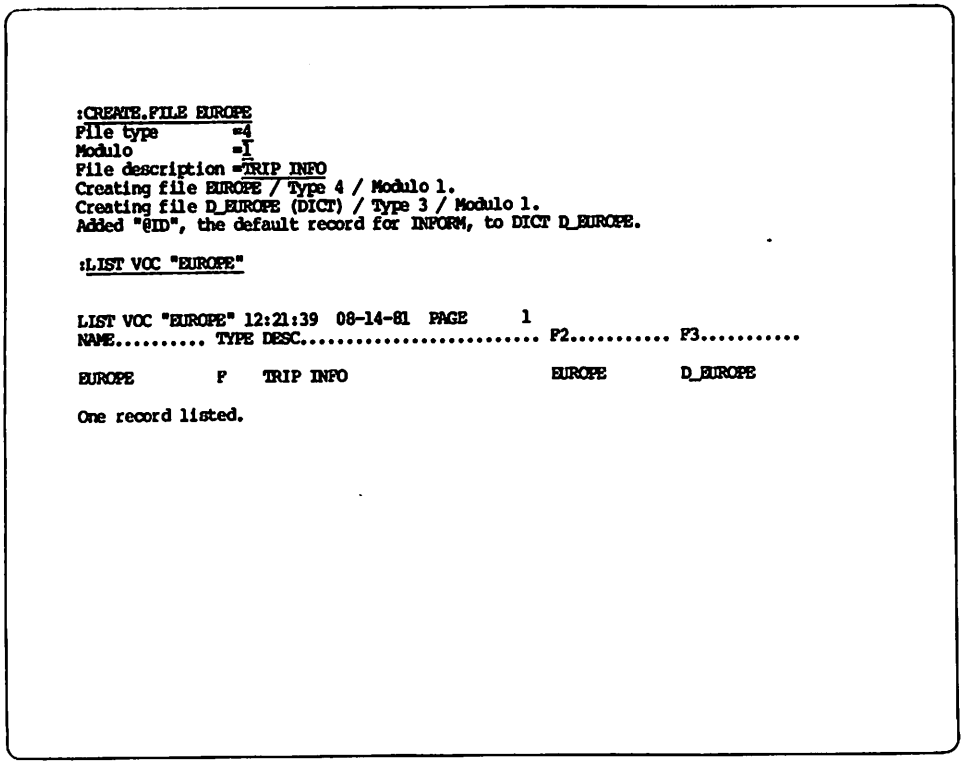
If you use the default form of CREATE.FILE to create both the dictionary and data files, the dictionary file will be set up with a default file type of 3 and a modulus of 1. Often, this type and size may prove incompatible with the type and amount of information that is eventually stored in the dictionary. If you anticipate a large number of entries for the dictionary file, that is, a large number of fields for the corresponding data file record, you may want to specify a larger modulus for the dictionary file. This can be done upon creation of the file by using the CREATE.FILE DICT format of the CREATE.FILE command. To change the dictionary file type or modulus at a later time, use the PERFORM RESIZE command, described in the PERFORM Reference Guide.

The VOC File Entry

Every INFORMATION file created in an INFORMATION account is described by an entry in the VOC file associated with that account. This entry is called a file definition (or description) entry. A file definition entry always has an F in Field 1 of the VOC file record to indicate that it is a file definition record. Depending on the form of the CREATE.FILE command used, the other fields in this file definition entry will vary.

Examples of VOC file entries created by the three different versions of the CREATE.FILE command are shown in Examples 2-2, 2-3, and 2-4.

Creating Both Data and Dictionary Files: Example 2-2 illustrates the creation of both a dictionary and data file using the first version of CREATE.FILE shown above.



```
:CREATE.FILE EUROPE
File type        =4
Modulo           =1
File description =TRIP INFO
Creating file EUROPE / Type 4 / Modulo 1.
Creating file D_EUROPE (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_EUROPE.

:LIST VOC "EUROPE"

LIST VOC "EUROPE" 12:21:39  08-14-81  PAGE     1
NAME..... TYPE DESC..................... F2..... F3.....

EUROPE         F    TRIP INFO                    EUROPE         D_EUROPE

One record listed.
```

Example 2-2. Creating Dictionary and Data Files

Notice that there are entries in both Field 2 and Field 3 of the VOC file entry. As always, INFORM assigns a default name to the dictionary file, using the convention D_filename, where filename is the name you entered during the CREATE.FILE sequence. In this case, the dictionary is named D_EUROPE. Dictionary files are always created with a file type of 3, as indicated by the explanatory message printed out by CREATE.FILE.

Creating a Dictionary File Only: When creating just a dictionary file, INFORM does not ask you for a file type or modulus, because the default values of 3 and 1 are automatically used. Notice that INFORM prefixes any name you give it with D_, which is the convention for indicating a dictionary file. Example 2-3 shows the process of using CREATE.FILE to create just a dictionary file.

```
:CREATE.FILE DICT EUROPE.DICT
Creating file D_EUROPE.DICT (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_EUROPE.DICT.

:LIST VOC "EUROPE.DICT"

LIST VOC "EUROPE.DICT" 15:55:33  08-17-81  PAGE     1
NAME..... TYPE DESC..... F2..... F3.....

EUROPE.DICT    F                                    D_EUROPE.DICT

One record listed.
```

Example 2-3. Creating a Dictionary File

There is no entry in Field 2 of the VOC file record because there is no data file associated with this dictionary.

To form an association between a dictionary file and some data file, edit the appropriate file definition entry in the VOC file with the command:

ED VOC dictionary.filename

dictionary.filename is the name of the dictionary file. Notice that you do not need the D_ prefix when specifying the name of the dictionary.

Then insert the name of the data file you want associated with this dictionary file in Field 2 of the file definition record. The first field in the record, when listed with the INFORMATION editor, contains the letter F and an optional description of the file.

Creating a Data File Only: Example 2-4 shows how to set up just a data file.

```
:CREATE.FILE DATA EUROPE.DATA
File type          =5
Modulo             =2
File description =DATA FILE
Creating file EUROPE.DATA / Type 5 / Modulo 2.

:LIST VOC "EUROPE.DATA"

LIST VOC "EUROPE.DATA" 15:58:43  08-17-81  PAGE     1
NAME..... TYPE DESC..... F2..... F3.....

EUROPE.DATA    F    DATA FILE                    EUROPE.DATA

One record listed.
```

Example 2-4. Creating a Data File

No dictionary file is created for this data file. In order to associate this data file with a dictionary, you'd have to edit the VOC file entry and put the name of the dictionary in Field 3 of the file definition record.

Note

For most purposes it is best to use the format of the CREATE.FILE command used in Example 2-2. Both parts of the file are then set up and all the necessary entries and pointers are taken care of at once so there's less chance of confusion. Used as in Example 2-2, CREATE.FILE prompts you for all the information it requires. Like all PERFORM commands, however, the CREATE.FILE verb accepts all of this information on the command line so you don't have to go through the prompting sequence.

Creating the STUDENTS File: The sample terminal session in Example 2-5 shows how the CREATE.FILE verb was used to create a sample file called STUDENTS. This file is used as an example throughout this book. The file uses social security numbers as keys (record IDs). Since this kind of key is numeric with separators (-), the file should be a Type 3 file. See the INFORMATION Highlights Companion if you need a review of file types or a quick summary of how to set up files.

```
:CREATE.FILE STUDENTS
File type          =3
Modulo             =1
File description =STUDENT NAMES AND GRADES
Creating file STUDENTS / Type 3 / Modulo 1.
Creating file D_STUDENTS (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_STUDENTS.
```

Example 2-5. Using the CREATE.FILE Verb

Events in File Creation

When a file is created using the default form of CREATE.FILE, two things happen automatically:

1. A pointer to the new file is entered in the VOC file in the form of a file definition record. Such a record is identified by an F in Field 1, and it contains the names of the data file and the dictionary file that describes it.

2. A data file with the name filename and a dictionary file with the name D_filename are created, where filename is the name specified on the CREATE.FILE command line.

Example 2-6 shows what the VOC file entry for the STUDENTS file looks like.

```
:LIST VOC "STUDENTS"

LIST VOC "STUDENTS" 14:02:27  06-25-81  PAGE     1
NAME..... TYPE DESC..................... F2..... F3.....
STUDENTS      F    STUDENT NAMES AND GRADES      STUDENTS      D_STUDENTS

One record listed.
```

Example 2-6. VOC Entry for STUDENTS File

After its creation, the new dictionary file for STUDENTS (D_STUDENTS) contains the information shown in Example 2-7.

```
:LIST DICT STUDENTS

LIST DICT STUDENTS 14:02:45  06-25-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@ID            D    0                    STUDENTS       10L    S

One record listed.
```

Example 2-7. Initial Contents of STUDENTS Dictionary File

Filename Synonyms

Because the physical representation of data in an INFORMATION file remains distinct from its logical structure, you can refer to the same data with more than one logical description. This means you can have more than one dictionary file for a single data file. In addition, more than one data file can be associated with a single dictionary. In effect, this makes filename synonyms possible. Since each association of a dictionary and data file must have a unique file definition entry, each different combination of data and dictionary file must have a different filename. Creating synonyms for filenames requires some adjustments to be made in your VOC file. These adjustments are outlined in the sample sequence below.

Sample Synonyms: Suppose you have an existing file definition entry in a VOC file that refers to the file STUDENTS. The data file is called STUDENTS and the dictionary file is called D_STUDENTS. If you wanted to create an alternate definition of the data in the STUDENTS file, you would have to create a new dictionary file. Suppose you wanted to call this new dictionary file STUDENT.COURSES. You can accomplish this by following these steps:

1. Use CREATE.FILE in the form shown here.

   CREATE.FILE DICT STUDENT.COURSES

2. Edit the VOC file entry that describes this file.

   EDIT VOC STUDENT.COURSES

3. Change Field 2 (which will be blank) to STUDENTS.

4. Use ENTRO to establish the contents of the dictionary file.

   ENTRO DICT STUDENT.COURSES

Now there are two different dictionaries for the same data file. To reference the data with the original dictionary, use the filename STUDENTS. To reference the data using the alternate definition, use the filename STUDENT.COURSES.

Defining Dictionary Contents

To define the contents of a dictionary file you can use either the INFORMATION Editor or the ENTRO processor. Use the INFORMATION Editor (ED) only if you know the types and formats of dictionary records. ENTRO, on the other hand, is an interactive entry tool that does not require you to know the exact format of a data file record. This means you don't have to know the names and locations of each field in the dictionary record. ENTRO tells you what the names are and what it expects you to enter for each field. To invoke ENTRO, type:

ENTRO DICT filename

where filename is the name of the data file you are describing. It is not necessary for the data file to exist in order to define a dictionary file for it.

ENTRO then displays:

```
FIELD=
```

which means that a field name is expected. If the field name you enter has not already been defined for this file, then ENTRO prompts for a descriptor type (TYPE) and an optional explanation (EXP):

```
New record
TYPE + EXP=
```

You must enter one of the following descriptor type codes in response to the TYPE + EXP= prompt:

- D — a data descriptor
- I — an I-descriptor
- PH — a user-defined phrase

The type code can be followed by an optional description of the field being defined.

Note

A special type of descriptor, called an X-item or X-descriptor, is reserved for users and is ignored by INFORM. This enables users to store certain kinds of information in the dictionary instead of the data file. X-descriptors cannot be created with ENTRO; you must use the INFORMATION Editor to do this. Put an X in Field 1 and put the the information you want stored in Field 2, 3, and so on. User-created X-descriptors can only be accessed via I-descriptor expressions and via INFO/BASIC programs, and are ignored by INFORM verbs. More references to X-descriptors can be found later in this section within the discussion of the TRANS function, and in Section 4, ENTRO KEYWORDS.

ENTRO's Dictionary Prompts

The list below shows all the prompts ENTRO displays when asking for details about a new descriptor. Unless otherwise indicated, a response is required for each one of these prompts. Optional prompts may be satisfied by a single carriage return.

Prompt	User Response
FIELD=	Enter field name.
New record	ENTRO message indicating creation of new record.
TYPE+EXP=	Enter D, I, or PH, plus optional description.
LOCATION=	Enter location, expression, or phrase items.
CONV=	Optional: enter INFO/BASIC conversion specification.
DISPLAY NAME=	Optional: enter display name for this field.
FORMAT=	Enter output format, e.g., 10T, 7R$###.##.
S/M=	Enter S for single, M for multivalued fields.
ASSOC=	Optional: used with multivalued fields only.

Sample Prompting Sequences: Example 2-8 shows the ENTRO prompts and user responses needed to define the LNAME (Last Name) descriptor in the STUDENTS file.

```
:ENTRO DICT STUDENTS
INFORM DICTIONARY DEFINITION  ENTRO.1  14:05:22 25 JUN 1981

FIELD=LNAME
New record
TYPE+EXP=D LAST NAME
LOCATION=1
CONV= (CR)
DISPLAY NAME=LAST NAME
FORMAT=15L
S/M=S
ASSOC=(CR)
```

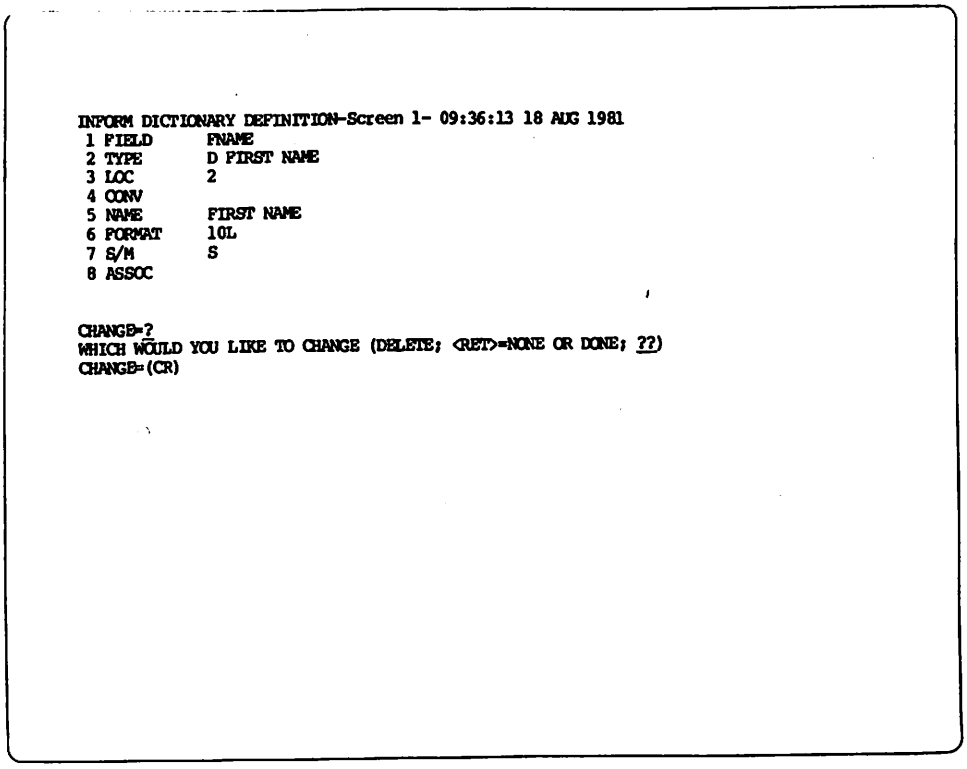
Example 2-8. Creating a Data Descriptor

Instant Help

A brief explanation of what is required as input to each of ENTRO's prompts can be had by typing a question mark (?) in response to any ENTRO prompt. The prompt will be redisplayed after the explanatory message is printed, as shown in Examples 2-9 and 2-10:

```
:ENTRO DICT STUDENTS
INFORM DICTIONARY DEFINITION  ENTRO.1  09:34:34 18 AUG 1981


FIELD=?
THE NAME OF THE FIELD  OR  END TO QUIT
FIELD=FNAME
New record
TYPE+EXP=?
ENTER THE LETTER D FOR A DATA FIELD, I FOR INFO EXPRESSION OR PH
FOR PHRASE FOLLOWED BY THE TEXTUAL EXPLANATION
TYPE+EXP=D FIRST NAME
LOCATION=?
THE NUMBER OF THE FIELD ON THE RECORD; THE EXPRESSION; THE PHRASE
LOCATION=2
CONV=?
ANY VALID INFORMATION CONVERSION IF REQUIRED
CONV=(CR)
DISPLAY NAME=?
NAME OF FIELD FOR DISPLAY IN INFORM REPORTS
DISPLAY NAME=FIRST NAME
FORMAT=?
LENGTH AND JUSTIFICATION FOR DISPLAY  (I.E. 20L)
FORMAT=10L
S/M=?
S(INGLE) VALUE OR M(ULTI) VALUE FIELD.
S/M=S
ASSOC=?
IF MULTIVALUED, THE NAME OF THE ASSOCIATIVE GROUP IF ANY
ASSOC=(CR)
```

Example 2-9. Getting Help in ENTRO



Example 2-10. Getting Help in ENTRO (Continued)

If no explanatory prompt exists for a field, ENTRO prints a message to that effect.

Modifying a Dictionary

A dictionary record can also be modified using ENTRO. New entries can be added any time and old ones can be deleted. To change an existing descriptor, enter the descriptor name in response to the FIELD= prompt. The entire record will then be brought up for you to modify. ENTRO has many special conventions that can be used in modifying dictionary file records. They are all explained in Section 3 under DATA ENTRY AND MODIFICATION.

Adding New Dictionary Entries: New I-descriptors and phrases can be added to a dictionary at any time without affecting the size of the data file. Such additions can impact the size and efficiency of the dictionary file. There are some tools available in INFORMATION to help you check the efficiency of the files in terms of organization and distribution of records. These tools are the GROUP.STAT, HASH.HELP, and HASH.TEST commands, which are documented in the PERFORM Reference Guide. Be sure to check the dictionary file occasionally with the GROUP.STAT command to ensure that it is properly organized. See Checking File Organization below.

Adding New Data Descriptors: When adding new data descriptors to a dictionary, the data file is apt to grow larger than you may have originally anticipated when you set it up. This is because the data record grows larger as more fields are added to it. This may require that the original modulus defined for the file may have to be changed to accommodate the larger average record size. Recall that the modulus is a way of indicating how many groups of records a file should have to be organized in the best manner possible. The most efficient file should have its records fairly evenly distributed over groups of approximately 1900 bytes each. If there are more than 1900 bytes per group, the file becomes more difficult to search. That is why it's important to keep the modulus large enough to accommodate a good record distribution of approximately 1900 bytes per group.

Checking File Organization: To determine whether the file is properly sized and organized, use the PERFORM GROUP.STAT command. It tells you how many bytes are in each group, how many records are in each group, the average number of bytes and records per group, and the standard deviation from the average group size. If the standard deviation is greater than about 10, use the HASH.HELP command to get a recommendation on proper file type and/or modulus. Each group should have approximately 1900 bytes.

Next use the HASH.TEST command to see how the file would be organized with the recommended modulus and/or file type. Use the RESIZE command to change either or both of the parameters once you've determined what they should be. All of these commands are PERFORM-level utilities and are described in the PERFORM Reference Guide.

Changing a Descriptor Record

Using ENTRO to change a field or two in a dictionary file record is simple. Example 2-11 shows how several fields in the default record ID descriptor, @ID, were changed to make them more meaningful. The TYPE field of the descriptor, which also contains a description (EXP) of the field, is changed to provide a better explanation of the record ID for this file. The display name of the @ID field is changed to SOC-SEC-NO to make it more meaningful as a key name. In addition, the output format of this field is changed to accommodate the maximum number of characters contained in the social security number field.

```
:ENTRO DICT STUDENTS @ID
INFORM DICTIONARY DEFINITION  ENTRO.1  14:03:57 25 JUN 1981


FIELD=@ID
INFORM DICTIONARY DEFINITION-Screen 1- 14:04:05 25 JUN 1981
 1 FIELD      @ID
 2 TYPE       D Default record id for Inform
 3 LOC        0
 4 CONV
 5 NAME       STUDENTS
 6 FORMAT     10L
 7 S/M        S
 8 ASSOC


CHANGE=2
TYPE+EXP=D SOCIAL SECURITY NO.
CHANGE=5
DISPLAY NAME=SOC-SEC-NO
CHANGE=6
FORMAT=11L
CHANGE=(CR)
```

Example 2-11. Making Changes With ENTRO

DICTIONARY FILE CONTENTS

Now that you've seen a brief overview of what a dictionary is and does, a more detailed description of its parts is necessary. The next portion of this section describes in detail the contents of a dictionary file. The following topics are covered:

- Data descriptor records
- The @ID record
- I-descriptor records
- Phrase records

For each of these record types, a general record format is given, showing the various fields in that record, what each is called, and what each field represents. A detailed field-by-field description follows this general record format for each of the four types of records.

DATA DESCRIPTORS

In the dictionary file record, a typical data descriptor record has a general format that can be described as shown in Table 2-1. The fields have been numbered as they would be if you displayed the contents of any data descriptor record with the INFORMATION Editor.

Note

When creating a data descriptor with ENTRO, you will notice that the field numbers shown here do not correspond exactly to the numbers used by ENTRO in its "change" sequence. This is because ENTRO makes up its own sequence numbers depending on the way the @ or @ENTRO phrases are defined, and depending on which fields you specify in the ENTRO sentence itself. If you specify that only certain fields be displayed, then the numbers will be different than they would be if all the field names listed in the @ENTRO or @ phrases were displayed. Nevertheless, the information in the first field of the record always corresponds to ENTRO's TYPE+EXP= prompt, the information in the second field corresponds to the LOCATION= prompt, and so forth. The field labeled "id:" is really Field 0 of the data record.

Table 2-1
Data Descriptor Record Format

Field Number	Field Contents	Description
id:	field.name	Identifies field being described.
001:	D [desc]	Type code and optional description.
002:	location	Number of field in record: 1, 2, 3, and so forth.
003:	conversion	Any legal INFO/BASIC conversion formula. Input for this field must be compatible with this formula, e.g., numeric conversions require numeric input.
004:	display name	Name to be displayed in place of field name indicated in the "id:" field.
005:	format	Any legal INFO/BASIC format, provided that the format starts with display length and justification.
006:	s/m	S for single value fields (one value per field per record) or M for multivalued fields (with one or more values per field per record).
007:	assoc	Optional association name; for multivalued fields only.
008: and on	Reserved	Don't touch!

These fields are described in detail on the following pages. The field numbers used correspond to the field numbers used by the Editor, not necessarily the field numbers used by ENTRO.

First, however, Example 2-12 shows two sample records as listed by the Editor. Notice the field numbers in the display. The descriptors are LNAME and FNAME and are from the STUDENTS file. LNAME was defined earlier in Example 2-8.

```
:ED DICT STUDENTS

File name = DICT  STUDENTS
Record name = LNAME
7 lines long.

---: P7
0001: D LAST NAME
0002: 1
0003:
0004: LAST NAME
0005: 15L
0006: S
0007:
Bottom at line 7
---: Q

File name = DICT  STUDENTS
Record name = FNAME
7 lines long.

---: P7
0001: D FIRST NAME
0002: 2
0003:
0004: FIRST NAME
0005: 15L
0006: S
0007:
Bottom at line 7
---:
```

Example 2-12. Dictionary Record Contents

Field Name Indicator — Field 0

The first field (Field 0) of a record indicates the name or key value by which the record is identified. This is the record ID (key) value for this descriptor. Every record name (field name) must be unique in a file. Duplicate record ID values are not allowed. In the above example, the field name values are LNAME and FNAME, respectively.

Type and Description Field — Field 1

Field 1 of the data descriptor record defines type code and description. The type code for a data descriptor is D. The optional description is a useful reminder of what this field contains. During data entry, you can type a question mark (?) in response to ENTRO's data entry prompt and this description will be displayed.

Location Field — Field 2

Field 2, the LOCATION field, contains a number like 1, 2, 3, etc., which indicates the relative position in the data record of the field being described. For instance, in the sample STUDENTS file, the field LNAME defines record position 1 in the data record, and the FNAME and AGE fields define positions 2 and 3. Figure 2-1 illustrates the relationship between the dictionary file and the data file record. The LOCATION field describes the relative position of a field within a record, and not the actual position.

Note

Do not assign the same field position to more than one data descriptor. Each data descriptor should describe only one field in a data file. This means the LOCATION field should have a unique value for each descriptor defined in a particular dictionary file.

Conversion Field — Field 3

The conversion field can contain any one of the legal conversion specifications that are available with the INFO/BASIC OCONV function. This specification is used to convert the stored data value to a desired output format. When using ENTRO to enter data for a field with a conversion specification, the data, assuming that it is compatible with the specification, will be converted to internal format using this formula. The effects of the conversion specification are obvious when the data is displayed. If the conversion specification is invalid, ENTRO flags this the first time you attempt to add data to this field. Assuming that a valid conversion has been specified, if the input data doesn't match the conversion specification, ENTRO will tell you about this also.

Note

The conversion specifications are put into effect only when data is entered with ENTRO. When entering data with the Editor, the conversion constraints will be ignored.

DICTIONARY FILE: D_STUDENTS

FIELD NAME	TYPE	LOCATION	CONVERSION	DISPLAY NAME	FORMAT	S/M	ASSOC
@ID (RECORD ID)	D	Ø		SOC-SEC-NO	11L	S	
LNAME	D	1		LAST NAME	15L	S	
FNAME	D	2		FIRST NAME	15L	S	
MI	D	3		MID.INIT.	2L#	S	
GRAD.YR	D	4		GRAD YEAR	4L	S	

Several Records From Dictionary File

DATA FILE: STUDENTS

	RECORD ID FIELD Ø	FIELD 1	FIELD 2	FIELD 3	FIELD 4
RECORD 1	131-43-5670	JARVIS	ADELLE		1983
RECORD 2	301-45-9817	MAYER	MARVIN	G	1985

Sample Records From Data File

Figure 2-1. Dictionary and Data File Layout for STUDENTS File

Types of Conversions: There are four categories of conversion specifications available:

- Masked Decimal Conversions (MD) — Formats numeric data for output with decimal points, commas and dollar signs.
- Date Conversions (D) — Converts internal date format to desired output format.
- Time Conversions (MT) — Converts internal time format to desired output format (hours, minutes, seconds, etc.).
- Number Base Conversions:
  - Binary (MB) — Transforms stored data values into their binary equivalents.
  - Hexadecimal (MX) — Transforms stored data values into their hexadecimal equivalents.
  - Octal (MO)— Transforms stored data values into their octal equivalents.

Masked Decimal Conversions: To display numeric values in decimal format, use the MD conversion specification of the OCONV function (INFO/BASIC). The basic format of the MD conversion specification is:

```
MDn[f][,][$]
```

The n is a required single digit (0-9) representing the number of digits in the data value that should appear to the right of the decimal point in the output. If you do not want a decimal point to appear in the output, specify n as 0.

The optional f argument represents the scalar factor, or position of the implied decimal point. If omitted, the decimal point will appear n places to the left of the last digit in the output.

If you want comma separators for thousands, millions, and so forth, use the comma (,) option.

Similarly, to include a prefix dollar sign ($) in the output, use the $ option.

For example, suppose you wanted to output a stored field with two digits to the right of a decimal point and a leading dollar sign. (The decimal point is not stored with the data.) Simply specify:

MD2$

in Field 3.

There are many other options available for masked decimal conversions. See the INFO/BASIC Reference Guide for details.

Note

ENTRO will automatically check input for any data descriptor with an MD indicator in Field 3 to make sure it contains numeric characters only. ENTRO also converts the data to internal format as dictated by the conversion specification.

Date Conversions: The general representation of a date conversion specification would be:

D [y][c][E]

y is an optional digit representing the number of digits to be output for the year. If not specified, the default value is four digits.

c is the optional character used to separate the month, day, and year. For example, c can be a hyphen (-) or a slash (/) character. When c is specified, the month will appear as a two-digit number (01-12). If c is not specified, then the month will be output as the three-character abbreviation for the month name, for example, NOV.

The E indicator causes the European date format to be used. European dates are expressed in the form:

day/month/year

The standard date format used by INFORMATION is:

month/day/year

Some examples of date formats and the output obtained by using these formats are shown here:

Formula	Internal Date	Output
D	4500	26 APR 1980
D2	4500	26 APR 80
D2/	4500	04/26/80

The form in which dates are stored internally is explained in the INFO/BASIC Reference Guide.

Time Conversions: Time conversions should be specified in the format:

MT[H][S][c]

The optional H indicates 12-hour time format, with AM or PM appended to the output as appropriate. The default is 24-hour format.

The S indicates that seconds should be included in the output. The default is to express only hours and minutes.

The c represents the optional separator character which is used to separate hours, minutes, and seconds. The default separator is the colon (:).

For example, if you specify the 12-hour format as in:

MTH

the internal time value of 10000 will be printed out as:

02:46AM

Number Base Conversions: To output numbers in hexadecimal, binary, or octal format, use the appropriate conversion codes indicated earlier. See the INFO/BASIC Reference Guide for additional details.

Display Name Field — Field 4

The display name field (Field 4) specifies an optional column heading for the descriptor. This name will be displayed at the terminal, or printed in a line printer listing whenever this field is referenced. If you don't specify a display name, the actual field name (in Field 0 of this record) will be used as the default column heading.

Multiline Headings: The display name (Field 4) can contain value marks (^253) to fold the heading name onto multiple lines, as shown below. The first example shows how a heading would be displayed without value marks and the second one shows the resulting display when a value mark (^253) is inserted between the two words in the name.

Entry in Field 4	Display
004: TYD TAXES	YTD TAXES
004: YTD^253TAXES	YTD TAXES....

Putting value marks into display name text can only be done with the INFORMATION Editor. ENTRO does not recognize the escape (^) character. Example 2-13 shows how value marks can be inserted into text within a record using the INFORMATION Editor.

```
:ED DICT STUDENTS COURSES
7 lines long.

---: P5
0001: D COURSES TAKEN
0002: 5
0003:
0004: COURSES TAKEN BY STUDENT
0005: 12T
---: PO 4
00 4: COURSES TAKEN BY STUDENT
---: C/ BY/^253BY/
0004: COURSES TAKEN]BY STUDENT
---: FILE
"COURSES" filed in file "DICT  STUDENTS".
```

Example 2-13. Creating Multiline Headings

The column heading altered in this example will now be displayed as:

COURSES TAKEN
BY STUDENT

Format Field — Field 5

Field 5 of each data descriptor record specifies the output format for the data associated with that descriptor. The output format indicates how the data will be displayed, or printed out. The output format specifies the length and justification for the output data. It may optionally specify a field padding character, a conversion formula, or a mask field. By specifying the output format in the dictionary file, rather than placing a restriction on the actual size of the field in the data file, the output format can be changed easily without affecting the data stored in the file. For instance, if you discover that you didn't allow enough room for a certain field to be printed out in its entirety, simply increase the field size specification in Field 5.

The general format for Field 5 should be:

field.size [bkgnd] just [conv][mask]

The arguments shown in this format are described in this list and in the text following it:

Argument	Meaning
field.size	Size (width) of the display field
bkgnd	Field-padding character (default is blank)
just	Justification (R, L, or T)
conv	Conversion expression (for output)
mask	Output format pattern

Field Size: The field.size argument specifies the width of the display field in which the value is to be printed out. A field size must be supplied in every output specification. A mask may also be supplied. (See below.) If the field value to be displayed is longer or shorter than the indicated field size, different things happen, depending on the type of terminal or line printer you are using.

- If the formatted data is smaller than the field size specification, the field will be padded with the background character. (The default is a blank.)
- If the data to be output is longer than the field size, the data will simply be folded over to the next line. This is the case on most terminals and line printers.

Field Padding Character: bkgnd indicates the special character to be used in field padding. This is optional. The default padding character is a blank. Padding is done when the field value to be printed out is smaller than the indicated field size.

Justification: The just argument indicates whether the field value should be left or right justified (if it's numeric) or treated as text. One of the following justifications must be specified in all formats:

L Left justification

R Right justification

T Text (No justification, see below.)

The Text (T) specification treats a block of text differently than an L or R specification. If the text contains embedded spaces, the output lines are broken up using the field size (field.size) as a guide, but only where a space occurs in the text line. This prevents word fragmentation. If there are no spaces in the text, the lines are broken up according to the number of characters specified in the field size argument. For example, if the field size was 10, there would be 10 characters printed per line.

Note

Strictly numeric data should always be specified with right justification (R). Numbers will then line up correctly for display and they will also sort properly. It looks much neater when doing totals and averages if the numbers are all lined up beginning with the rightmost digit.

Strictly alphabetic data, like text and strings containing only characters from the 64-character ASCII subset, should always be specified with T (text) justification. This ensures that they will be properly formatted for output and that they can be sorted properly also.

Data made up of alphanumeric characters should always be specified with left justification (L) to make them line up properly.

Conversion: The conversion argument permits dollar signs and commas to be added in numeric items when they're displayed. It is used for output formatting only. Conversions are specified in this general form:

n[$][,][Z]

n is the number of fractional digits in the value. The other format characters are optional. If used, they should be typed with no intervening spaces. The $ character causes a dollar sign to be output in front of the field value. Inclusion of a comma in the conversion expression causes commas to appear after every three digits in the output. The Z character indicates zero suppression and causes leading 0's to be suppressed in the output.

For example, the TUITION field in the STUDENTS file has a format specification of:

```
10R2$,
```

which indicates that the output field is a maximum of 10 digits, including a leading dollar sign and a decimal point. The R specifies right-justification which is customary for numeric fields, and the 2 indicates that there are two digits to the right of the decimal point. A comma will be inserted if there are more than three digits to the left of the decimal point. Using this format specification, the value 4775 will be output as:

```
$4,775.00
```

Mask: The mask field allows literals to be intermixed with digits in a specified pattern. During output, mask formatting is performed before any of the other output options. If the data to be formatted is longer than the specified mask, the data will be truncated. After the mask portion of the formatting has been done (or if none was specified), the resulting string will be adjusted to the field.size specification.

For example, the MI (Middle Initial) field from the STUDENTS file has a format specification of:

```
2L#.
```

which specifies that the output value for this field will contain two digits, one of which is a number (represented by #) and the other of which is a period. An example of the mask portion of a format specification is:

```
###-##-##
```

which could be part of the format for a part number specification. If the value 5383283 were printed out according to the above mask, the output would appear as:

```
538-32-83
```

Single/Multivalue Indicator — Field 6

Field 6 of each data descriptor record indicates whether the data is single-valued or multivalued. Field 6 is also known as the S/M (single/multivalue indicator) field. If you expect only one value for a particular field in a record, then the S/M indicator should be defined as "S".

For example, the AGE field in an employee file would be a single-valued field because each person should have only one age at any given time. This means there is one AGE value for each record.

Multivalued Fields: Multivalued fields are used when you need to have more than one data value in a particular field. For example, in the STUDENTS file, the GRADES field would be multivalued, since each student is likely to have more than one grade. Each record in such a file would have one or more values for the GRADES field.

The S/M indicator should be set to M if the field is to have multiple values. This tells INFORM that it should look for value marks in this field. A value mark is defined as CHAR(253) which appears as a left brace ({) on terminals and line printer listings. A value mark is inserted between each individual value in a multivalued field. (They are put in automatically by ENTRO.)

Multivalued field items may themselves be composed of smaller entities called subvalues. Subvalues are indicated by the subvalue mark, which is equivalent to a CHAR(252). This prints as a | character.

Specifying a field as S (single-valued) does not prevent you from putting more than one value in this field, but it makes it easier for other users to tell which fields are multivalued and which are not, simply by looking at Field 6. This also tells ENTRO to look for value marks when dealing with the field, which improves processing efficiency.

Association Field — Field 7

If a data field is specified as multivalued, it may also belong to an association. An association relates two or more multivalued fields so that for every value in one field, there is a companion value in each of the associated fields. Whenever you add, change, or delete a data value from one field in an association, you must do the same to the corresponding values in the associated fields.

To form an association, you must create a phrase that names the fields that you want associated. Field 7 of each descriptor whose name appears in such a phrase should in turn contain the name of the phrase.

Sample Association: In the STUDENTS file, data about courses and course grades should be stored together in some way. Since most students will take more than one course, the COURSES and GRADES fields should be multivalued fields. It is also important that each course have the proper grade associated with it. Thus, an association should be created between the COURSES and GRADES fields. The phrase used to associate them could be called COURSE.GRADES. Example 2-14 shows how to create the phrase called COURSES.GRADES which links these two fields.

```
FIELD=COURSES.GRADES
New record
TYPE+EXP=PH COURSE & GRADE ASSOC.
LOCATION=COURSES GRADES
CONV=
DISPLAY NAME=
FORMAT=
S/M=
ASSOC=
INFORM DICTIONARY DEFINITION-Screen 1- 14:16:16 25 JUN 1981
 1 FIELD      COURSES.GRADES
 2 TYPE       PH COURSE & GRADE ASSOC.
 3 LOC        COURSES GRADES
 4 CONV
 5 NAME
 6 FORMAT
 7 S/M
 8 ASSOC


CHANGE=
```

Example 2-14. Creating a Phrase for Associations

Examples 2-15 and 2-16 show how these multivalued descriptors were defined. The COURSES field lists all the courses that a student takes and the GRADES field lists all the grades that correspond to these courses. Both descriptors are defined with an M in Field 6. The name of the association phrase, COURSE.GRADES, is put in Field 7 of both the COURSES and GRADES descriptors.

```
FIELD=COURSES
New record
TYPE+EXP=D COURSES TAKEN
LOCATION=5
CONV= (CR)
DISPLAY NAME=COURSES TAKEN BY STUDENT
FORMAT=12T
S/M=M
ASSOC=COURSE.GRADES
INFORM DICTIONARY DEFINITION-Screen 1- 14:10:46 25 JUN 1981
 1 FIELD      COURSES
 2 TYPE       D COURSES TAKEN
 3 LOC        5
 4 CONV
 5 NAME       COURSES TAKEN BY STUDENT
 6 FORMAT     12T
 7 S/M        M
 8 ASSOC      COURSE.GRADES

CHANGE=(CR)
```

Example 2-15. Creating a Multivalued Descriptor (COURSES)

```
FIELD=GRADES
New record
TYPE+EXP=D GRADE FOR COURSE
LOCATION=6
CONV=
DISPLAY NAME=GRADE FOR COURSE
FORMAT=5L
S/M=M
ASSOC=COURSES.GRADES
INFORM DICTIONARY DEFINITION-Screen 1- 14:12:54 25 JUN 1981
 1 FIELD      GRADES
 2 TYPE       D GRADE FOR COURSE
 3 LOC        6
 4 CONV
 5 NAME       GRADES FOR COURSES
 6 FORMAT     5L
 7 S/M        M
 8 ASSOC      COURSES.GRADES

CHANGE=(CR)
```

Example 2-16. Creating a Multivalued Descriptor (GRADES)

Multivalued Descriptors: Example 2-17 shows all the multivalued descriptors in the STUDENTS file dictionary. Multivalued descriptors are indicated by an M in the S/M (single and multivalue field indicator) field.

```
:LIST DICT STUDENTS WITH SM EQ M

LIST DICT STUDENTS WITH SM EQ M 16:33:22  08-15-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

COURSES        D    5                   COURSE NUMBER   12T    M  COURSES.GR
                                                                  ADES
GRADES         D    6                   COURSE GRADE    5L     M  COURSES.GR

2 records listed.
```

Example 2-17. Multivalued Fields in STUDENTS File

Data in Multivalued Fields: Example 2-18 shows how data in multivalued fields is displayed. The two fields listed here are the COURSES and GRADES fields from the STUDENTS file. These two fields are members of the association called COURSES.GRADES.

```
:LIST STUDENTS LNAME COURSES GRADES WITH LNAME EQ "MAYER"

LIST STUDENTS LNAME COURSES GRADES WITH LNAME EQ "MAYER" 14:44:41  06-25-81  PA
GE     1
SOC-SEC-NO.    LAST NAME.....    COURSE NUMBER    GRADE FOR COURSE

301-45-9817    MAYER             CS105            B
                                 CS108            B+
                                 CS110            A-

One record listed.
```

Example 2-18. Data in Multivalued Fields

@ID DESCRIPTORS

When a dictionary is first created, the only record it contains is the @ID record. This record is created automatically by INFORM and has the default format shown in Table 2-2. Again, the field numbers shown in this table correspond to those used by the INFORMATION Editor.

Table 2-2
@ID Record Format

Field Number	Field Contents	Description
id:	@ID	Identifies record ID.
001:	D [desc]	Type code and default description.
002:	0	Record ID values always define Field 0.
003:	blank	No conversion specified: user may supply one if desired.
004:	filename	Filename becomes default display name.
005:	format	Default output format is 10L for Type 2-13 files, and 32L for Type 1 files.
006:	S	Must be a single-valued field.
007:	blank	No associations for single-valued fields.

You can change the items in Fields 3, 4, and 5 of an @ID record. In general, Fields 0 - 7 of any descriptor can be modified by the user and all other fields must be left alone.

Note

If you change the name of the @ID record, which is a perfectly legitimate thing to do, avoid changing Field 2, the LOCATION field, which is set to 0 by default.

A Sample @ID Record: Example 2-19 shows what a new dictionary file looks like just after it is created. This particular example shows the default @ID descriptor for the STUDENTS file as listed with ENTRO. Notice that the field numbers used by ENTRO are different from those used by the Editor. The net effect, though, is that the name of the descriptor will be the first field listed. The default description:

Default record id for Inform

is used for every @ID descriptor. The default description, as well as the default display name, format, and record name, can be changed by the user, as mentioned earlier.

```
1 FIELD     @ID
2 TYPE      D Default record id for Inform
3 LOC       0
4 CONV
5 NAME      STUDENTS
6 FORMAT    10L
7 S/M       S
8 ASSOC
```

Example 2-19. An @ID Descriptor

I-DESCRIPTORS

I-descriptors describe information that does not exist in the data file, but which can be derived from existing information in one or more data files. I-descriptors, also called I-items, are identified by a type code of I in Field 1 of a dictionary record.

I-descriptors may include arithmetic operations, logical operations, string manipulations, INFO/BASIC functions, external subroutine calls, and a special process called file translation. All of these items are discussed later in this book.

Once an I-descriptor is defined in a dictionary, it may be used in exactly the same manner as are data field descriptors, that is, for listing, selecting, sorting, and so forth. I-descriptors can also define synonyms for fields in the data file that are already described by data descriptors. For this use, Field 2 of the I-descriptor record should contain the name of the data descriptor used to define this data field.

Format of an I-descriptor Record

I-descriptor records make use of their fields somewhat differently than data descriptors. Field 2 defines the actual expression to be evaluated, rather than a relative field location in a data file record. Fields 8 on must not be modified by the user, as they contain the compiled object code of the expression in Field 2. See Compiling I-descriptors below. The general format of an I-descriptor record is shown in Table 2-3 below. The field numbers are shown as they appear when listed with the Editor.

Table 2-3
I-descriptor Record Format

Field Number	Field Contents	Description
id:	field.name	Name of I-descriptor
001:	I [desc]	Type code and optional description
002:	expression	Expression to be evaluated
003:	conversion	Conversion specification (optional)
004:	name	Optional display name
005:	format	Format specification (required)
006:	s/m	Single or multivalued field (required)
007:	assoc	Association name (optional)
008 thru 015		Reserved
016:	code	Compiled object code (Don't touch!)
017:	time	Time descriptor was last compiled (Time is in internal format)
018:	date	Date (in internal format) last compiled
019:	code	Compiled object code (Don't touch!)
020 and on		Reserved

Compiling I-descriptors

The I-descriptor expression in Field 2 must be checked for syntax errors before it can be evaluated to a particular value. The syntax checking process is called compilation because the expression is translated into executable code as a result of a completed, error-free syntax check. This executable code is called compiled object code.

Syntax checking includes verifying that descriptor names are legal, and that the specified operations can be done on the data in the indicated fields. See I-descriptor Expressions later in this section. The syntax checking of an I-descriptor can be done in one of two ways:

- By using the CD verb, followed by the descriptor name.
- By default, when it is referenced by any INFORM verb.

The dictionary expression compiler is invoked to compile the expression, either directly by using CD or indirectly by the default method.

The CD Verb: There are two forms of the CD verb. This form:

```
CD filename
```

compiles all the I-descriptors in the dictionary of the indicated file.

The second format:

```
CD filename descriptor-name1 [descriptor-name2] ...
```

compiles only the named descriptors in the dictionary of the indicated file. Any errors found during compilation will be reported to the user and compilation of that particular I-descriptor will not be completed. Compilation of any remaining I-descriptors in the file, however, is not affected by the failure of any previous I-descriptor compilation.

Default Compilation: Any I-descriptor expression in a file can be compiled when it is referenced by any INFORM verb if it has not already been compiled. However, using the CD verb is a good idea because it lets you know if there are any syntactical or logical problems in your I-descriptor expression before you attempt to use it.

Note

The object code generated by compiling an I-descriptor may not exceed 188 characters. If it does, a message to that effect will be displayed, and no code will be created for that I-descriptor. You can convert a lengthy expression which exceeds this limit into a cataloged INFO/BASIC subroutine, which can then be called by the I-descriptor. To keep the object code small, use short descriptor names and variables when possible. (See also @-variables later in this section.)

Sample Compilation: Example 2-20 displays show what happens when you use the INFORMATION Editor to look at an I-descriptor that has not yet been compiled.

```
:ED DICT STUDENTS NGRADES
This Type "I" Descriptor must be compiled before use.
7 lines long.

----: P7
0001: I CONVERTS LETTERS TO NUMBERS
0002: TRANS(NUMGRD,GRADES,NUM,"X")
0003:
0004: NUMERIC GRADE
0005: 2L1
0006: M
0007:
Bottom at line 7
----: Q

:CO STUDENTS NGRADES
Compiling NGRADES = TRANS(NUMGRD,GRADES,NUM,"X")
Compiled  TRANS(NUMGRD,GRADES,NUM,X)
```

Example 2-20. Compiling an I-Descriptor

When you edit the record after compilation, the date and time of the last compilation will be printed out in place of the "must be compiled before use" message. It is not advisable to list lines 8 onward at a terminal as the results are often unpredictable. However, the contents of a compiled I-descriptor record can be safely viewed on all terminals if you use the ^ (up-arrow) command of the Editor before displaying the record. (See the INFORMATION Editor Reference Guide for details.)

Sample I-descriptors

All I-descriptors must have an I in the TYPE field and the expression to be evaluated in Field 2 (the LOCATION field). The expression is entered in response to ENTRO's LOCATION= prompt. This field is used for the expression because a field position is irrelevant for an I-descriptor. There is no field value associated with it in the data file. The value for an I-descriptor is obtained dynamically by evaluating the expression.

Table 2-4 shows some examples of I-descriptors. The first two yield numeric values that are formatted according to the conversion specifications indicated. MD indicates masked decimal conversion. The numeric arguments indicate the number of digits that should appear to the right of the decimal point.

MD0 indicates that no digits are to appear after a decimal point; the decimal point would not even be output in this case. MD2 indicates that two digits should be printed to the right of the decimal point. (Masked decimal conversion is explained more fully in the INFO/BASIC Reference Guide.)

Table 2-4
Sample I-descriptors

Name	Type	Expression	Conversion
TOTAL.AREA	I	LAND.AREA + WATER.AREA	MD0
VALUE	I	QTY*COST	MD2
GRAPH.VALUE	I	STR('*',VALUE/5)	50L

The first I-descriptor, TOTAL.AREA, defines the value obtained by adding the data in the LAND.AREA field to the WATER.AREA field. Both LAND.AREA and WATER.AREA could be either data descriptors or I-descriptors.

The second descriptor, VALUE, defines the result of multiplying the QTY field by the COST field, both of which could be either data or I-descriptors.

The third descriptor, GRAPH.VALUE, an I-descriptor, produces a string of asterisks with a length determined by the expression 'VALUE/5'. Sample output from this evaluation is shown later in this section under Functions in Expressions.

I-descriptors in the STUDENTS File: Example 2-21 lists the I-descriptors in the dictionary of the STUDENTS file. Each one will be described in more detail below.

```
:LIST DICT STUDENTS WITH TYPE EQ I

LIST DICT STUDENTS WITH TYPE EQ I 10:07:10  10-20-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

COUNT.COURSES  I    COUNT(COURSES,                      2R
                    CHAR(253))+1
GRADS          I    IF GRAD.YR =                        12L
                    "1981" THEN
                    "ACTIVE" ELSE
                    "INACTIVE"
GPA            I    SUM(NGRADES)/(C       GPA           5R2    S
                    OUNT(GRADES,@VM
                    )+1)
GT.3.0         I    SUM(SUBR('-GTS'       GRADES ABOVE B  2L
                    , NGRADES,
                    REUSE("3.0")))
NGRADES        I    TRANS(NUMGRAD,G       NUMERIC GRADE   2L1    M
                    RADES,NUM,"X")
TUITION        I    SUM(TRANS(COSTS MDO   TUITION COSTS   10R2$, S
                    ,COURSES,3,"X")
                    );IF @1 THEN
                    @1+25 ELSE 0

6 records listed.
```

Example 2-21. I-descriptors in DICT STUDENTS

I-DESCRIPTOR EXPRESSIONS

Field 2 of every I-descriptor record contains an expression. An expression defines how the value for this I-descriptor should be derived. The sample I-descriptors shown above illustrate several dictionary expressions. These expressions follow rules that are very similar to the expression rules of INFO/BASIC. I-descriptor expressions are also called dictionary expressions in INFORMATION. The possible components of expressions, and the rules for creating them, are discussed next.

I-descriptor Expression Components

Expressions are composed of elements that get operated on or manipulated in some way, and operators that specify what is to be done to those elements, that is, how they are to be operated on. Table 2-5 lists all the possible kinds of elements that can appear in an I-descriptor expression, and Tables 2-6, 2-7, and 2-8 list all the kinds of operators that can be used to manipulate these elements.

Expression Elements

I-descriptor expressions can contain the kinds of elements listed in Table 2-5. These elements can be combined with operators (described below) to form I-descriptor expressions.

Table 2-5
I-descriptor Expression Elements

Elements	Description
field.names	Names of descriptors in this dictionary
numeric constants	Numeric values like 20, 5, 10.3
string literals	Must be in quotes ("Hello")
@variable	Internal variables called "@-variables"
functions	Any legal INFO/BASIC function, like STR or FIELD, or an INFORM function like TOTAL or TRANS
(expression)	An expression enclosed in parentheses

Sample I-descriptor Elements: Here is an example of an I-descriptor expression which contains several of the above elements:

```
TOTAL(COST*QTY)
```

Starting from the left, the word TOTAL indicates the INFORM TOTAL function, used in I-descriptors to add up all the values that occur in a given descriptor or expression involving two or more descriptors. (TOTAL is described fully under TOTAL Function in I-descriptors, later in this section.) The words QTY and COST are field names. They are combined by the * operator, which indicates multiplication, and are enclosed in parentheses to indicate that the TOTAL function should operate on the value produced by multiplying the values in these two fields together.

Operators in Expressions

According to the rules of INFO/BASIC, the above elements of Table 2-5 may be combined with arithmetic, relational, and logical operators. Most of these operators can also be used in INFO/BASIC expressions. INFORM has some keywords that perform the same types of operations as a number of these operators. This has been indicated where possible in the tables that follow.

Arithmetic Operators: The arithmetic operators which can be included in INFORM I-descriptor expressions as well as in INFO/BASIC expressions are shown in Table 2-6.

Table 2-6
Arithmetic Operators

Operator	Meaning
+	Unary plus
-	Unary minus
**	Exponentiation
^	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction
:	Concatenation
CAT	Concatenation

An example of an I-descriptor expression that contains an arithmetic operator is:

```
(QTY*COST)/5
```

The values in the QTY and COST fields are multiplied together as indicated by the asterisk (*), which is the multiplication operator. The / operator indicates division and indicates that the entire expression should be divided by 5, which is a numeric constant representing the divisor for the parenthetical expression.

Relational Operators: Expressions may be joined by any of the relational operators defined in INFO/BASIC. Some of these operators are also defined as INFORM keywords. In general, INFORM keywords cannot appear in I-descriptor expressions unless they are identical to an INFO/BASIC relational operator. Use only those operators listed in Table 2-7 when defining I-descriptors.

Table 2-7
Relational Operators for I-descriptors

Operator	Meaning
EQ =	Equal to
NE # >< <>	Not equal to
> GT	Greater than
< LT	Less than
=> >= GE	Greater than or equal to
=< <= LE	Less than or equal to
#>	Not greater than
#<	Not less than
MATCHES	String value matches pattern

For instance, the relational operator =>, which means "greater than or equal to", is used in the expression:

```
IF GRAD.YR => "1981" THEN "ACTIVE" ELSE "INACTIVE"
```

All the values in the GRAD.YR field that are greater than or equal to 1981 will satisfy the comparison, and the expression will be returned as "ACTIVE". For GRAD.YR values that are less than or equal to this year value the expression will be returned as "INACTIVE".

Logical Operators: The Boolean operators, AND and OR, are also supported. They are listed in Table 2-8 and are used in the form:

```
expr AND expr

expr OR expr
```

where expr represents an arithmetic or relational expression that can be evaluated to true (1) or false (0). Both AND and OR can be combined in the same expression.

Table 2-8
Logical (Boolean) Operators

Operator	Meaning
AND	Evaluates to 1 if both expressions are true.
OR	Evaluates to 1 if either expression is true.

Conditional Expressions

Conditional expressions are also legal in I-descriptors. Such expressions are commonly referred to as "IF...THEN..." expressions, or "conditionals." Conditionals are evaluated to true or false, and the action taken depends on the outcome of the evaluation. Such expressions are written in the form:

```
IF expr1 THEN expr2 ELSE expr3
```

The expressions expr2 and expr3 represent the possible values which the I-descriptor will be assigned based on the value of expr1.

Note

The ELSE clause is required in all INFORM conditionals.

Example of a Conditional Expression: Several of the expressions shown previously made use of the IF...THEN...ELSE conditional construction. Here is another I-descriptor that uses a conditional expression:

```
IF GRAD.YR EQ 1981 AND GPA GT 3.2 THEN "HONORS" ELSE "AVG"
```

If the value of the GRAD.YR field equals 1981 and the value for the GPA field in the same record is greater than 3.2, then the expression evaluates to true, and the value "HONORS" is returned. However, if both conditions are not met within a single record, the I-descriptor expression will evaluate to false and the ELSE clause will be executed. The value of the expression will be returned as "AVG" when this happens. The AND operator requires that both conditions must be met in order for the expression to evaluate to true.

References in Expressions

I-descriptors refer to the contents of a field in a record by the name of the field, not by its relative location number within the record. Any field name referenced in an I-descriptor expression must be defined in the dictionary at the time the I-descriptor expression is compiled. Whenever an I-descriptor is referenced within an expression, its name is replaced by the corresponding expression specification, enclosed within parentheses (). Thus, valid expression structure is preserved. The expansion process is repeated until no further I-descriptor names appear in the expression being compiled.

TOOLS FOR I-DESCRIPTOR EXPRESSIONS

There are many tools available to help you define powerful I-descriptor expressions. Combined cleverly, they can eliminate the need to write separate programs to do things like calculating totals and sines, performing character-to-decimal conversions, converting uppercase to lowercase, extracting certain characters from strings, trimming trailing or leading blanks from strings, and a host of other useful things. Some of the tools are easier to use than others. You'll find yourself using a small subset of them in nearly every I-descriptor you create. Others are designed for more specialized purposes and will only be useful occasionally.

The next part of this section introduces and explains how to use the following expression tools:

- Substring Extraction in Expressions
- Functions in Expressions
- @-variables in Expressions
- Compound Expressions
- File Translation (TRANS)
- Subroutine Calls in Expressions
- Multivalue-handling Subroutines
- The TOTAL Function in Expressions

Substring Extraction in Expressions

Substring extraction in I-descriptors allows you to strip off parts of expression elements, field values, literals, numeric constants, and so forth, and use these pieces in evaluating the entire expression. Any of the elements which appeared in Table 2-5 may be immediately followed by one of these expressions:

Expression	Description
[ex1,ex2]	Extract ex2 characters beginning at ex1.
[ex1,ex2,ex3]	Extract the ex2 occurrence of the string separated by the character ex1 and extract ex3 number of fields.

The items ex1, ex2, and ex3 represent any legal dictionary expressions.

Substring Extraction Example: The following is an example of the use of substring extraction within an I-descriptor expression. In an inventory file which lists various articles of clothing, their prices, composition, and so forth, the record ID field is the item number. It is a five-character field, of which the first two are numbers to indicate fabric, the third indicates style, and the last two indicate what type of garment it is.

This descriptor searches the @ID fields, takes the last two characters off of each @ID value and compares them to the string PT, which represents pants. If a match is found, the value PANTS will be printed out for this field in the display. Otherwise, two asterisks will be printed.

The substring extraction expression specifies that two characters are to be stripped off the @ID value, beginning with the fourth character from the beginning of the string:

```
IF @ID[4,2] EQ "PT" THEN "PANTS" ELSE "**"
```

The quotes in this I-descriptor expression are required.

Functions in Expressions

Most of the functions available in INFO/BASIC can also be used in dictionary expressions. The use of these functions and their arguments is the same as in INFO/BASIC. The arguments of these functions may consist of any legal expression.

Table 2-9 shows all the functions which are supported for use in I-descriptor expressions. The variables x, a, b, c, and d represent any legal descriptor expressions as described above.

Table 2-9
Functions Used in I-descriptor Expressions

ABS(x)
ASCII(x)
ATAN(x)
CHAR(x)
COL1()
COL2()
CONVERT(a,b,c)
COS(x)
COUNT(x,x)
DATE()
DOWNCASE(x)
EBCDIC(x)
EXP(x)
EXTRACT(a,b,c,d)
FIELD(a,b,c)
FIELD(a,b,c,d)
FMT(a,b)
ICONV(a,b)
INDEX(a,b,c)
INT(x)
LEN(x)
LN(x)
LOWER(x)
MATCHFIELD(a,b,c)
MOD(a,b)
NOT(x)
NUM(x)
OCONV(a,b)
PWR(a,b)
RAISE(x)
REUSE(x)
RND(x)
SEQ(x)
SIN(x)
SPACE(x)
SQRT(x)
STR(a,b)
SUM(x)
TAN(x)
TIME()
TIMEDATE()
TRIM(x)
TRIMB(x)
TRIMF(x)
UPCASE(x)

Functions Available in INFORM Only

The functions listed in Table 2-10 are defined for use only in INFORM I-descriptors, and not in INFO/BASIC programs. All the other functions listed in Table 2-9, with the exception of CONVERT, are intended for use in INFO/BASIC programs as well as in I-descriptors, and are explained in the INFO/BASIC Reference Guide.

Table 2-10
Functions for I-descriptors Only

Function	Description
DOWNCASE	Converts all uppercase letters to lowercase.
LOWER	Converts field marks () to value marks ({), value marks to subvalue marks (|).
RAISE	Is the reverse of LOWER.
UPCASE	Is the reverse of DOWNCASE.

The RAISE and LOWER functions are intended for use with multivalued fields. The format for their use is:

```
RAISE(expr)
LOWER(expr)
```

where expr is a string that names a field or represents a legal expression that can be evaluated to yield some string value.

RAISE and LOWER Conversions: The RAISE function will convert any of the five system delimiter characters that appear in expr to the next level of delimiter character. These conversions include:

IM	CHAR(255)	to	IM	CHAR(255) (no conversion)
FM	CHAR(254)	to	IM	CHAR(255)
VM	CHAR(253)	to	FM	CHAR(254)
SM	CHAR(252)	to	VM	CHAR(253)
TM	CHAR(251)	to	SM	CHAR(252)

The LOWER function will convert any of the five system delimiter characters which occur in string expr to the next lower level of delimiter. The conversions are:

IM	CHAR(255)	to	FM	CHAR(254)	
FM	CHAR(254)	to	VM	CHAR(253)	
VM	CHAR(253)	to	SM	CHAR(252)	
SM	CHAR(252)	to	TM	CHAR(251)	
TM	CHAR(251)	to	TM	CHAR(251)	(no conversion)

Function Example: Here is an example of the UPCASE function. A file contains information which is stored in a mixture of uppercase and lowercase letters. When searching for particular values, it is bothersome to have to remember which things are in which case. An I-descriptor can do this for you. For example, to find a record with a certain last name value, define a descriptor called UPNAME as:

```
UPCASE(LNAME)
```

When the sentence:

```
LIST BIRTHDAYS WITH UPNAME EQ DOUROS
```

the record in the file whose LNAME field can be converted to DOUROS via this function will be retrieved. The actual value of the LNAME field for this record is Douros, not DOUROS, but the UPNAME expression converts the lowercase letters to uppercase.

Nonsupported Functions

The following INFO/BASIC functions are not available for use within dictionary expressions:

CONVERT a TO b IN c Use this form instead: CONVERT(a,b,c)

DELETE(x,x,x,x)

INMAT()

INSERT(x,x,x,x,x)

REPLACE(x,x,x,x,x)

More Examples of Functions

The next two I-descriptors use functions to produce a graph. The VALUE I-descriptor record looks like this:

```
001:  I
002:  QTY*COST
003:  MD2
004:
005:  10R
006:  S
007:
```

The GRAPH.VALUE I-descriptor record is defined as:

```
001:  I
002:  STR('*',VALUE/5)
003:
004:  GRAPH.OF.VALUE
005:  50L
006:  S
007:
```

The STR function takes the value produced by evaluating the I-descriptor VALUE, divides it by 5, and prints out that number of stars (*) to form a graph. The output below shows what happens when the contents of the VALUE and GRAPH.VALUE descriptors are displayed.

```
LIST INV VALUE GRAPH.VALUE  11:15:19  10-20-78  PAGE      1
INV..... VALUE..... GRAPH.OF.VALUE.....................................

10001         150.00 ******************************
10002          33.00 ******
10003          37.50 *******
10004         250.00 **************************************************
```

Another useful function is the COUNT function. To find the number of courses a students has taken, use the COUNT function on the COURSES field. Since this is a value-marked field, the number of entries in it can be determined by counting the number of value marks, or { characters, which occur in the field, and adding 1 to this number. Since the first value in the field is not preceded by a value mark, it is necessary to add 1 to the COUNT value in order to achieve a true count.

An I-descriptor for the STUDENTS file called COUNT.COURSES could be defined as:

```
COUNT(COURSES, CHAR(253))+1
```

There are other ways of counting the number of occurrences of a certain string in a value-marked field and are described later in this section.

@-variables in Expressions

Within an expression, it is possible to have special variables for which values are dynamically substituted when that expression is evaluated. Such variables, called @-variables, are preceded by the @ symbol. They can represent certain internal information, like time and date. They can also be assigned the results of other simple expressions within a compound I-descriptor expression. Compound expressions are described later in this section under Compound Expressions.

INFORM-defined @-variables: The @-variables listed in Table 2-11 are defined in INFORM and can be included in I-descriptor expressions.

Table 2-11
Special INFORM @-variables

Variable	Defines
@DATE	Internal date (does not change during verb execution)
@DAY	Day of month from @DATE (range 1 - 31)
@FILE.NAME	Current filename
@FM	A field mark character: CHAR(254)
@ID	Record ID of the current record
@IM	An item mark character: CHAR(255)
@MONTH	Current month number (1-12; does not change during verb execution)
@RECORD	Entire current data record
@SM	Subvalue mark character: CHAR(252)
@TIME	Unformatted internal time (does not change during verb execution)
@TM	Text mark character: CHAR(251)
@USER.NO	User number (determined from hardware configuration)
@VM	Value mark character: CHAR(253)
@YEAR	Current two-digit year number (does not change during verb execution)

@-variable Examples

To illustrate how some of these functions can be used in I-descriptor expressions, look at the GPA descriptor from the STUDENTS file. This descriptor makes use of the SUM and COUNT functions, both of which are listed in Table 2-9, above. It also uses one of the special @-variables, @VM, shown in Table 2-11. The SUM and COUNT functions are not to be confused with the INFORM level SUM and COUNT verbs. (Functions and verbs are not the same thing. Functions are described in this section and verbs are described in Section 3.)

The SUM function combines numeric values in a multivalued field to yield a single field value. If the field being summed contains subvalues, which are separated by subvalue marks (|), the subvalues between each set of value marks ({) are summed, and the result of the SUM function will be a multivalued field. The result will contain value marks, but no subvalue marks. It's important to note this because the result is not a single field value when the multivalued field contains subvalues.

The COUNT function counts the number of occurrences of a specified string within a given field. This function is useful for counting string occurrences in single-valued or multivalued fields. In this example, COUNT is used to determine the number of values in the GRADES field, which is a multivalued field.

The GPA I-descriptor is defined as:

```
SUM(NGRADES)/(COUNT(GRADES,@VM)+1)
```

How GPA Works: The I-descriptor GPA uses the SUM function to add all the grade point values defined by the I-descriptor NGRADES. The result of adding the values in this multivalued field is a single numeric value. To obtain each student's grade point average, this value is then divided by the number of grades which have been summed.

The number of grades which have been summed is obtained with the COUNT function. It counts the number of occurrences of the value mark, represented by the special @VM variable, that appear in the GRADES field. It then adds 1 to this number to obtain the actual number of entries in the field. Adding 1 is necessary because the first value in a multivalued field is not preceded by a value mark. Thus, there is always one less value mark than actual values. The GPA descriptor is defined as shown in Example 2-22.

```
:LIST DICT STUDENTS GPA

LIST DICT STUDENTS GPA 10:02:48  10-20-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

GPA            I    SUM(NGRADES)/(C      GPA            5R2    S
                    OUNT(GRADES,@VM
                    )+1)

One record listed.
```

Example 2-22. I-Descriptor Using SUM and COUNT

Evaluating the GPA Descriptor: Example 2-23 shows a sample of the data obtained by evaluating the GPA I-descriptor expression:

```
:LIST STUDENTS FNAME LNAME COURSES.GRADES GPA

LIST STUDENTS FNAME LNAME COURSES.GRADES GPA 10:08:45  10-20-81  PAGE     1
SOC-SEC-NO. FIRST NAME..... LAST NAME..... COURSE NUMBER COURSE GRADE GPA..

301-45-9817 MARVIN          MAYER          CS101         B             3.33
                                           MG101         B+
                                           CS673         A-
412-05-7716 ELAINE          GOUDREAUX      CS105         B             3.43
                                           CS108         B+
                                           CS216         A
131-43-5670 ADELLE          JARVIS         CS101         B             3.00
                                           CS301         A
                                           CS579         A-
                                           CS673         D
                                           CS673         B+
111-45-5566 STEVEN          MCLEAN         MG101         B             3.33
                                           MK101         A
                                           EN201         B
343-05-9988 MARTHA          GRANIFF        SP101         B             3.43
                                           MK101         A
                                           MK108         B+
141-05-9988 SUSAN           KATZ           CS101         A             4.00
                                           MK101         A
                                           MG101         A
Press <NEW LINE> to continue...
LIST STUDENTS FNAME LNAME COURSES.GRADES GPA 10:08:45  10-20-81  PAGE     2
SOC-SEC-NO. FIRST NAME..... LAST NAME..... COURSE NUMBER COURSE GRADE GPA..

                                           MK107         A

6 records listed.
```

Example 2-23. Evaluating GPA

Compound Expressions

Dictionary expressions may be simple expressions or compound expressions. Compound expressions consist of two or more simple expressions separated by semicolons (;). Compound expressions may be used to increase execution efficiency and to improve the readability of an expression. The syntax of a compound expression is:

```
expr ; expr ; expr ; ...
```

Simple expressions are evaluated from left to right, and the resulting value for the entire compound expression is the value of the final simple expression. The value of each simple expression may be referenced by the following @-variables:

@	Value of the previous simple expression
@1, @2, @3...	Value of a specific simple expression

Simple expressions are numbered from left to right as they occur in the compound expression. The @-variable @ (with no number) refers to the previous simple expression to the left.

Compound Expression Example: Another I-descriptor defined for the STUDENTS file serves as an example of a compound expression. The TUITION descriptor figures out the cost of each course the student is taking and adds up the total, then adds in a $25.00 tuition fee. First, however, it checks to see whether or not the student is taking any courses, because if there are no courses listed, no tuition fee should be reported. Here is what the I-descriptor expression in Field 2 of the TUITION record looks like:

```
SUM(TRANS(COSTS,COURSES,3,"X");IF @1 THEN @1+25 ELSE 0
```

The SUM function is used to add up all the values returned from the COSTS file. (The TRANS function is described later on, and its exact purpose here is not important for this discussion.) It simply looks up the fee for each course that is listed in the COURSES field of the STUDENTS file. The result of summing this expression is then assigned to the special @-variable, @1. The IF...THEN...ELSE conditional clause checks the value of @1 to see if it is 0 or not. If the value is 0, the expression evaluates to false, and the whole thing is returned as 0. This happens when the student is not taking any courses. When the expression evaluates to true, that is, when @1 is anything but 0, the entire expression is returned as the value @1+25, which is the sum of all courses being taken plus the $25.00 tuition fee.

Note

Compound expression structures should not be nested. A nested compound dictionary expression might result from the expansion of one expression that refers to another compound I-descriptor expression by its descriptor name. This type of nested reference is illegal, as shown in the following examples.

Illegal Compound Expressions: If a compound expression contains the name of an I-descriptor which is itself a compound expression, INFORM will be unable to expand the expression. For example, suppose there were an I-descriptor called NET.PROFIT, defined as:

```
SUM(PROFIT - FIRST.COST);IF FIRST.COST GE NET.SALES THEN 'IN RED'
ELSE @1
```

Further, the FIRST.COST descriptor is another I-descriptor that defines this compound expression:

```
MAN.COST+TAX+LAND.FEE;IF MAN.COST GT 300 THEN MARKUP ELSE @1
```

The initial expression is illegal. INFORM cannot substitute the FIRST.COST expression into the first one properly, which produces an expression that is too complex to evaluate.

File Translation (TRANS)

The INFORM TRANS function allows you to extract information from other data files. This makes it possible to fetch field values, or entire records from some other data file and use them in evaluating some I-descriptor expression in the current file.

In order to do this, you must supply TRANS with the name of the file from which you want to extract the information. Think of this file as the "translate" file. You must also supply either the name of a field in the current file that contains the same values as the record ID in the translate file, or supply an ID expression which will produce values equivalent to the record ID values of the translate file.

Then you must specify the name or number of the field which you want retrieved from the translate file, or, an expression which involves a field name or number from the translate file. The entire record can also be extracted if desired, using a special @-variable called @RECORD. The results of the translation process are monitored with a special control code feature, which tells TRANS what to do in case of an unsuccesful translation.

There are several forms of the TRANS function available, depending on what information you intend to retrieve from the translate file.

Retrieving Fields From Other Data Files: To return field values from records in another data file, use the following TRANS format:

TRANS(trans.filename, id.expr, field.name, code.expr)

The trans.filename argument is the name of the translate file, and id.expr represents an expression that evaluates to record ID values in the translate file. Based on these record ID values, certain records will be selected from the translate file, and information from these records will be returned to the current I-descriptor for use in evaluating the I-descriptor expression.

The field.name argument represents the name of a field in the translate file which you want returned by this translate operation. field.name should be a data descriptor only. The records from which field.name is extracted will have record ID values that match those produced by id.expr.

The code.expr argument tells TRANS what to do if the translate operation is not successful. Control codes are described in Table 2-13.

Retrieving an Entire Record: To return the entire record from the translate file during a TRANS operation, use either of these formats:

TRANS(trans.filename, id.expr, @RECORD, code.expr)

or

TRANS(trans.filename, id.expr, -1, code.expr)

The arguments used here are identical to those used in the above format, and are described in Table 2-12. The special @-variable, @RECORD, or its equivalent, -1, tells TRANS to return the entire data record whose record ID value matches id.expr.

Retrieving Information From a Dictionary: You can retrieve information from a dictionary file instead of a data file by specifying the dictionary file as the translate file. Use either the DICT keyword, followed by the translate filename, or D_trans.filename, where trans.filename is the name of the translate file. The formats are:

TRANS(DICT trans.filename, id.expr, fieldno, code.expr)

or

TRANS(D_trans.filename, id.expr, fieldno, code.expr)

One way of using the format just shown is to retrieve data stored in X-type descriptors in the specified file dictionary. X-type descriptors are also called X-items. X-type descriptors, as mentioned previously in this section, are reserved by INFORMATION for the user to do with as desired.

TRANS Function Arguments: The arguments shown in the previous TRANS formats are described in Table 2-12.

Table 2-12
TRANS Function Arguments

Argument	Meaning
trans.filename	Actual name of translate file as it appears in the user's VOC file. It cannot be an expression or variable. This file must be in the user's VOC file both when the expression is compiled and when the expression is used.
fieldno	Actual field number of some field in the dictionary file which is being used as the translate file.
field.name	Actual field name in the translate file. Must not be an expression or variable. This field name must be defined in the dictionary of the translate file when the TRANS expression is compiled. This field must be a D-type descriptor, and cannot be an I-descriptor.
id.expr	Any expression that evaluates to an ID value of some record in trans.filename. The expression may simply be the name of a field in the current file or an expression involving fields in the current file.
@RECORD	Variable indicating that the entire record from trans.filename should be returned. Only a record whose record ID value matches the value of id.expr will be returned.
code.expr	A control code expression, as defined in Table 2-13.

Table 2-13
TRANS Function Control Codes

Code	Meaning
code = "C"	Translate if possible, otherwise use the original record ID if the specified record doesn't exist or if the specified field is null.
code = "V"	Verify successful translation; otherwise, print an error message and set result to NULL if the specified record does not exist or if the specified field is null.
code = "X"	Translate if possible, otherwise set result to NULL if the specified record does not exist or if specified field is null.

If the translation function is successful, then the value of the function will be either the specified field within the record whose record ID value matches that of id.expr, or the entire record whose record ID value matches id.expr.

If id.expr turns out to be multivalued, then the translation will be performed for each record ID value, and the result of the function will also be multivalued.

If id.expr is multivalued and the field being retrieved is also, a string of multiple values will be returned for each multivalued field. When the specified field in the record of the translate file is multivalued, the entire string of multiple values in this field will be returned exactly as they appear in the file.

TRANS Function Example: In the STUDENTS file, the NGRADES I-descriptor translates the alphabetical grades into numeric grades using the TRANS function. The translate operation takes each alphabetic grade and looks it up in the NUMGRD file. NUMGRD contains a list of letter grades, which are record ID values, and another field which contains the numeric values which correspond to these letter grades. For each letter grade from the STUDENTS file that matches a record ID value in the NUMGRD file, a numeric equivalent is returned in the NGRADES I-descriptor. Since the GRADES field is multivalued, the NGRADES I-descriptor will be returned as a string of multivalues.

Example 2-24 shows how the data is stored in the NUMGRD file, and Example 2-25 displays the NGRADES descriptor, which uses the NUMGRAD file in its translate operation.

```
:LIST NUMGRAD NUM

NUMGRAD...    NUM

P             0.0
A             4.0
A-            3.7
B+            3.3
B             3.0
B-            2.7
C+            2.3
C             2.0
C-            1.7
D             1.0

10 records listed.
```

Example 2-24. Data in NUMGRAD File

```
1 FIELD      NGRADES
2 TYPE       I CONVERTS LETTERS TO NUMBERS
3 LOC        TRANS(NUMGRD,GRADES,NUM,"X")
4 CONV
5 NAME       NUMERIC GRADE
6 FORMAT     2L1
7 S/M        M
8 ASSOC

CHANGE:
```

Example 2-25. I-Descriptor with TRANS Function

Subroutine Calls in Expressions

Cataloged INFO/BASIC subroutines may be called from dictionary expressions with the SUBR function. (The term cataloged means that the INFO/BASIC subroutines have been compiled and made available for general use on the INFORMATION system. See the INFO/BASIC Reference Guide for information.)

The syntax of the SUBR function is:

SUBR(name.expr, arg.expr, arg.expr, ...)

where name.expr is any expression which evaluates to the name of the subroutine to be called and arg.expr is any expression to be evaluated and passed to the called subroutine. As many as 254 arguments may be defined per subroutine call.

Here are some rules and tips regarding the use of the SUBR function:

- Arguments may be passed to the subroutine as needed.
- The SUBR function must specify exactly the same number of arguments as the called subroutine. (This includes the name.expr argument.)
- Upon return from the subroutine, the value of the first argument in the argument list will become the value of the dictionary expression.
- The called subroutine must return a single argument which in fact could be multivalued. The value in this argument becomes the value of the dictionary expression that used the SUBR function.
- Up to 254 arguments may be defined per subroutine call.
- Any called subroutine must be a globally cataloged routine.
- The first argument of the called subroutine must be a variable which will contain the value returned by the subroutine.

Subroutine Example: Here is an I-descriptor expression that calls a globally cataloged subroutine called EXTEND:

```
001:  I
002:  SUBR("*EXTEND",COST,QUANTITY)
003:  MD2$
004:
005:  10R
006:  S
007:
```

The text of the cataloged INFO/BASIC EXTEND subroutine is:

```
SUBROUTINE EXTEND(RESULT,X,Y)
RESULT = X * Y
RETURN
END
```

The value contained in the variable RESULT will be the value of the I-descriptor EXTEND. This subroutine has been globally cataloged so that it can be used in I-descriptor expressions. The asterisk indicates that it is a globally cataloged routine. In the I-descriptor expression, the subroutine name is prefixed with an asterisk (for example, *EXTEND) to indicate that EXTEND is a globally cataloged routine. Cataloging is discussed in the PERFORM Reference Guide.

Another Subroutine Example: An I-descriptor that makes use of the special @USER.NO @-variable might be:

```
IF @USER.NO = 3 OR @USER.NO = 5 THEN SALARY
ELSE SUBR("*SECURITY.VIOLATION")
```

If this item were named ANNUAL.SALARY, it would be possible to restrict unauthorized users from access to the SALARY field. Users on terminal ports 3 and 5 (presumably in the payroll office) could view an employee's salary only, while any other attempted use would be handled by the subroutine SECURITY.VIOLATION. Subroutines are identified by the keyword SUBR, and are explained later in this section under Subroutine Calls in I-descriptors.

Multivalue-handling Subroutines

Special subroutines are provided by INFORMATION to define expressions that operate on fields which contain multivalues. These subroutines perform many of the same operations that are normally handled by standard functions when evaluating single-valued fields. Multivalue subroutines are necessary when operating on multivalued fields, because the standard operators and functions described previously in Tables 2-7 and 2-9 simply treat a multivalued field as a single string. They do not recognize value or subvalue marks. The multivalue-handling subroutines in Table 2-14 do recognize value marks and are able to process these values on an individual basis.

Table of Function/Subroutine Correspondence

Table 2-14 shows all the standard single value functions and the multivalue subroutines that correspond to them. In Table 2-14, m and ml represent multivalued fields, s and sl represent single-valued fields, and p1, p2, and so forth, represent single-valued parameters.

Table 2-14
Single Value Functions and Multivalue Subroutines

Single Value Function	Multivalue Subroutine
s:sl	SUBR('-CATS',m,ml)
s:pl:sl	SUBR('-SPLICE',m,pl,ml)
CHAR(s)	SUBR('-CHARS',m)
COUNT(s,pl)	SUBR('-COUNTS',m,pl)
s EQ sl	SUBR('-EQS',m,ml)
s NE sl	SUBR('-NES',m,ml)
s LE sl	SUBR('-LES',m,ml)
s LT sl	SUBR('-LTS',m,ml)
s GT sl	SUBR('-GTS',m,ml)
NOT(s)	SUBR('-NOTS',m)
FIELD(s,pl,p2[,p3])	SUBR('-FIELDS',m,pl,p2,p3)
FMT(s,pl)	SUBR('-FMTS',m,pl)
ICONV(s,pl)	SUBR('-ICONVS',m,pl)
INDEX(s,pl,p2)	SUBR('-INDEXS',m,pl,p2)
LEN(s)	SUBR('-LENS',m)
NUM(s)	SUBR('-NUMS',m)
OCONV(s,pl)	SUBR('-OCONVS',m,pl)
SEQ(s)	SUBR('-SEQS',m)
STR(s,pl)	SUBR('-STRS',m,pl)
SPACE(s)	SUBR('-SPACES',m)
s[pl,p2]	SUBR('-SUBSTRINGS',m,pl,p2)

The single value functions always return a single value, even when a multivalued field is supplied as a parameter. The multivalue subroutines always return a string of multivalues whenever a multivalued field name is supplied for m. This string of multivalues, also called a list, contains the results obtained by performing the function on each value of m.

Note

Subroutines that operate on two multivalued fields at the same time result in a single multivalued field. The actual operation is performed on each pair of associated values: value 1 with value 1, value 2 with value 2, etc. If the number of values differs, the missing values are treated as nulls, zeroes or spaces (blanks), depending on the data and the operation involved. When single-valued fields are used in place of multivalued arguments, they are treated as multivalued fields with only the first value defined.

Multivalue Subroutine Example

Suppose you wanted to find the number of grades above a B that a student has on record in the STUDENTS file. To do this, you have to search the NGRADES field which contains the numeric equivalents of the letter grades. A grade of B corresponds to 3.0 in the NUMGRAD file. Since NGRADES is a multivalued field, the multivalue subroutine '-GTS' must be used to determine how many values in the field are greater than 3.0.

The I-descriptor expression might look like this:

```
SUM(SUBR('-GTS', NGRADES, REUSE("3.0")))
```

The -GTS subroutine will compare the contents of NGRADES to the value 3.0 and will return a string of 0's and 1's, depending on which values match (1) or don't match (0) the search criterion (3.0). SUM simply adds up all the 0's and 1's, yielding a sum which represents the number of values that met the search.

REUSE must be used to apply the same value, in this case 3.0, to all members of the multivalued field NGRADES so a comparison can be made. If REUSE is not used, the value 3.0 will only be applied to the first member of the multivalued field. Using it guarantees that all members of the multivalued field will be compared to the same criterion.

Another Multivalue Subroutine Example

Suppose a file called OPTIONS contains information about certain option numbers that are contained in another file called NUMBERS. The NUMBERS file defines an I-descriptor called GET.DESC which fetches descriptions of the option numbers from the OPTIONS file using the TRANS function. The record ID entries of the OPTIONS file are equivalent to the record ID entries in the NUMBERS file, except that they are each prefixed with the string OPT*. The record ID field of the NUMBERS file is called OPTION.NO. The record ID values in the OPTIONS file which are needed in the translate operation can be derived by evaluating the following expression:

```
"OPT*":OPTION.NO
```

The OPTION.NO field is defined in the NUMBERS file and is a multivalued field. The record ID expression (id.expr in the TRANS function format) tells INFORM to prefix the string OPT* to each value in the OPTION.NO field and use the resultant value as a record ID value to obtain the record information needed from the OPTIONS file.

The Wrong Way: You might think that a statement of the form:

```
TRANS(OPTIONS,"OPT*":OPTION.NO, DESC, "X")
```

might work. However, OPTION.NO is a multivalued field, and the expression as stated would only prefix the string OPT* to the first value in this field. The translation operation would thus work only for the first value in this field, and would fail for the remainder of the option numbers. Instead you must use the multivalue handling subroutine -CATS, which is capable of prefixing the same string to each member of a multivalued field. Because OPT* must be prefixed to each value in the OPTION.NO field, use the REUSE function.

The Right Way: Thus, the correct form of the expression stated earlier would be:

```
TRANS(OPTIONS,SUBR("-CATS",REUSE("OPT*"),OPTION.NO),DESC,"X")
```

This expression would correctly return all the literal descriptions that correspond to the record ID values which are obtained by using the subroutine -CATS and the function REUSE.

The TOTAL Function in Expressions

The TOTAL function, abbreviated T, is most useful in preparing reports that contain numeric data that must be summed, averaged, and otherwise manipulated. It accumulates running totals on numeric fields so they can be used in evaluating I-descriptors on breakpoint lines. Such intermediate totals cannot be printed out or used on detail lines, but can only be used to calculate the value for an I-descriptor on the breakpoint line. Breakpoints are indicated in an INFORM sentence with the keyword BREAK.ON, followed by the name of the field at which breakpoints should occur. See Section 4 for information on BREAK.ON and all other INFORM keywords.

TOTAL Is Used With CALC: The TOTAL function is only used in conjunction with the CALCULATE (synonym is CALC) keyword in a LIST, SORT, or similar INFORM sentence that includes breakpoints. The special ability of TOTAL to accumulate separate running totals for each numeric field with which it is specified in an expression, can only be seen when the I-descriptor is CALCULATEd on a breakpoint line. For instance, if the I-descriptor MARKUP is defined as:

```
TOTAL(LAND.COST)/TOTAL(PRICE)*100
```

it would only be possible to obtain subtotals for the MARKUP fields on breakpoint lines if it were used in this type of sentence:

```
LIST GARMENTS BY FABRIC BREAK.ON FABRIC LAND.COST PRICE CALC MARKUP
```

Every time the value of the FABRIC field, which is the breakpoint field, changes, a new subtotal value for MARKUP will be displayed on the breakpoint line.

The expression will not be evaluated on a breakpoint line if the INFORM sentence specifies the MARKUP descriptor without the CALC keyword.

How TOTAL Works: The TOTAL function does two things. First, it tells the expression processor to maintain a running total of some expression on detail lines. Second, it tells the expression processor to reference those accumulated totals when evaluating the expression on a breakpoint line. Evaluations to be done on breakpoint lines are indicated by the keyword CALCULATE in an INFORM sentence. (CALCULATE is a keyword and cannot be used inside an I-descriptor expression. It can only be used in INFORM sentences.)

Note

It is not necessary for the field names referenced in an I-descriptor expression to be restated in the INFORM sentence.

Where TOTALs Are Available: On a detail line, the value of an expression containing the TOTAL function will be the same as if the TOTAL function had not been specified. That is, the accumulated total is not available on detail lines but is only available on breakpoint lines. The actual values of accumulated TOTALs are printed only on breakpoint lines.

When a descriptor expression is CALCULATEd on a BREAK.ON line, any field name or expression that is not an argument of a TOTAL will refer to the current value of the field name or expression. In the case of a field name, the current value on a breakpoint line is the same as the value on the last detail line.

Nesting TOTALs: As a general rule, TOTAL functions should not be nested. Nesting is generally the result of one I-descriptor TOTALing another I-descriptor that also contains TOTAL within its own expression specification. This can lead to interpretation problems by the expression compiler and should be avoided by using compound expressions instead. For example, the nested expression:

```
TOTAL(TOTAL(COST) * TOTAL(QUANTITY))
```

could be correctly expressed using compound expressions and @-variables, as shown here.

```
TOTAL(COST); TOTAL(PRICE); TOTAL(@1*@2)
```

When specifying expressions that contain the TOTAL function, it is important to remember the equivalencies listed in Table 2-15 to ensure arithmetically accurate results.

Table 2-15
TOTAL Equivalencies

TOTAL(A+B)	is equivalent to	TOTAL(A) + TOTAL(B)
TOTAL(A-B)	is equivalent to	TOTAL(A) - TOTAL(B)
TOTAL(A*B)	is not equivalent to	TOTAL(A) * TOTAL(B)
TOTAL(A/B)	is not equivalent to	TOTAL(A) / TOTAL(B)

Expressions Using TOTAL: The following is an example of an I-descriptor using the TOTAL function. The actual processing performed on detail lines and on CALCULATE (breakpoint) lines can be shown with the following equivalent INFO/BASIC code.

Assume that the I-descriptor expression is:

```
TOTAL(QTY*(PRICE-COST)) / TOTAL(QTY*PRICE) * 100
```

On detail lines the following statements would be equivalent:

```
field = QTY*(PRICE-COST) / QTY*PRICE * 100

TEMP1 = TEMP1 + QTY*(PRICE-COST)

TEMP2 = TEMP2 + QTY*PRICE
```

On the CALCULATE (breakpoint) line the equivalent calculation would be:

```
field = TEMP1 / TEMP2 * 100

TEMP1 = 0

TEMP2 = 0
```

The preceding statements are somewhat simplified since the actual calculations maintain running totals for each possible breakpoint (BREAK.ON) level, plus the grand total. (The BREAK.ON keyword is described in Section 4 under REPORT OPTIONS.)

A TOTAL Example: Examples 2-26 and 2-27 show how the TOTAL function can be used. The first display in Example 2-26 shows the dictionary and data file contents of an inventory file, INV.

```
:LIST DICT INV

LIST DICT INV 09:34:31  09-08-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@ID            D    0                   INV             10L    S
DESC           D    1                   DESCRIPTION     20L    S
COST           D    2              MD2  COST            8R     S
PRICE          D    3              MD2  PRICE           8R     S
QTY            D    4                   QUANTITY        4R     S
MARGIN2        I    TOTAL(QTY*(PRIC MD1P MARGIN          6R     S
                    E-COST))/TOTAL(
                    QTY*PRICE)*100
MARGIN         I    TOTAL(QTY*(PRIC MD1OP MARGIN         6R     S
                    E-COST))/TOTAL(
                    QTY*PRICE)*100
VALUE          I    COST * QTY     MD2                   10R    S
@              PH   DESC COST PRICE      THIS IS DISPLAY
                    QTY VALUE            NAME
                    MARGIN

9 records listed.

:LIST INV

LIST INV 09:34:35  09-08-81  PAGE     1
INV..... DESCRIPTION..... COST.... PRICE... QUANTITY VALUE..... MARGIN

10001    MOTOR              15.00    25.00       10     150.00   40.0
10002    TRANSFORMER         5.50     9.50        6      33.00   42.1
10003    CABINET             2.50     4.00       15      37.50   37.5
10004    AMPLIFIER         125.00   225.00        2     250.00   44.4

4 records listed.
```

Example 2-26. INV Dictionary and Data File

Following this, the individual totals for each of the fields in the file are displayed using the SORT verb. Lastly, the VALUE column is totalled and the MARGIN over all the possible COST, PRICE, and QTY values is calculated using the CALCULATE keyword.

```
:SORT INV DESC TOTAL COST TOTAL PRICE TOTAL QTY TOTAL VALUE TOTAL MARGIN

SORT INV DESC TOTAL COST TOTAL PRICE TOTAL QTY TOTAL VALUE TOTAL MARGIN
 09:34:38  09-08-81  PAGE     1
INV..... DESCRIPTION..... COST.... PRICE... QUANTITY VALUE..... MARGIN

10001      MOTOR              15.00    25.00       10     150.00   40.0
10002      TRANSFORMER         5.50     9.50        6      33.00   42.1
10003      CABINET             2.50     4.00       15      37.50   37.5
10004      AMPLIFIER         125.00   225.00        2     250.00   44.4
                             ======   ======   ======   ========   ====
                             148.00   263.50       33     470.50  164.0

4 records listed.

:SORT INV DESC COST PRICE QTY TOTAL VALUE CALCULATE MARGIN

SORT INV DESC COST PRICE QTY TOTAL VALUE CALCULATE MARGIN 09:34:43  09-08-81  P
AGE     1
INV..... DESCRIPTION..... COST.... PRICE... QUANTITY VALUE..... MARGIN

10001      MOTOR              15.00    25.00       10     150.00   40.0
10002      TRANSFORMER         5.50     9.50        6      33.00   42.1
10003      CABINET             2.50     4.00       15      37.50   37.5
10004      AMPLIFIER         125.00   225.00        2     250.00   44.4
                                                       ==========
                                                         470.50   42.4

4 records listed.
```

Example 2-27. Using The TOTAL Function

PHRASES

Phrases are included in a file dictionary by specifying the type code PH in Field 1 of a descriptor record. Enter a PH in response to the TYPE + EXP= prompt of ENTRO.

Structure of a Phrase Record

Each phrase record is stored with the information shown in Table 2-16. As in previous record formats, the field numbers correspond to those used by the INFORMATION Editor.

Table 2-16
Phrase Record Format

Field Number	Field Contents	Description
id:	phrase name	Name of phrase
001:	PH [desc]	Type code and optional description
002:	phrase contents	Actual contents of phrase
003 - 007	blank	N/A

Field 2, phrase contents, can contain descriptor (field) names, selection criteria, sort options, and keywords. In other words, the contents of a phrase can be any part of a legal INFORM sentence that does not contain an INFORM verb.

Sample Phrase Record: Example 2-28 is a listing of the COURSE.GRADES phrase record from the STUDENTS file. Note that Fields 4 through 8 are blank. You saw how this phrase was defined using ENTRO in Example 2-13.

```
1 FIELD     COURSES.GRADES
2 TYPE      PH COURSE & GRADE ASSOC.
3 LOC       COURSES GRADES
4 CONV
5 NAME
6 FORMAT
7 S/M
8 ASSOC
```

Example 2-28. Phrase Definition Record

Default INFORM Phrases

There are three default phrases recognized by INFORM:

- @
- @ENTRO
- @LPTR

The @ and @LPTR phrases govern the default descriptor names used in terminal display and line printer listings respectively. These phrases name the fields in a file that are to be displayed or printed out when the user doesn't specify any field names in an INFORM sentence.

The @ Phrase: The INFORM SORT and LIST verbs look for a phrase named @ when no field or expression names are mentioned in a sentence and when the LPTR keyword is not used. (LPTR directs the output from a sentence or command to the line printer. It is described in Section 4.) The names specified in the @ phrase are then used in the SORT or LIST operation.

When invoked for data entry, ENTRO first looks in the dictionary for the @ENTRO phrase so it knows what fields to prompt for input. The @ENTRO phrase is described below. If the @ENTRO phrase does not exist, ENTRO then looks in the dictionary file to see if the user has created an @ phrase. If so, ENTRO uses the descriptor names listed in this phrase to prompt for field values.

Note

If you want an @ phrase, you have to create it yourself. (Use ENTRO or the Editor to do this.) Be sure to name it @ so INFORM recognizes it and uses it correctly. If you create an @ phrase after an @ENTRO phrase has been created, the @ phrase will be ignored by ENTRO and will be used only by LIST and SORT.

The @LPTR Phrase: Similarly, when you want a report sent to the line printer (LPTR), the SORT and LIST first look for a phrase called @LPTR. If one is not found, the verbs then look for the @ phrase. Again, if you want a list of default field names to use in printing the results of executed INFORM sentences, you must create an @LPTR phrase in the appropriate file dictionary.

The @ENTRO Phrase: In the absence of a user-created @ phrase, the @ENTRO phrase is automatically created by ENTRO when invoked for the first time to add data to a new file. The @ENTRO phrase lists all the data (D-type) descriptors defined in the associated dictionary file at the time that ENTRO is invoked.

@ENTRO and Data Security: If new data descriptors are added to the dictionary after the @ENTRO phrase is created, you will have to update the @ENTRO phrase to include these new data descriptor names. If the new names are not added to the @ENTRO phrase, you will never be prompted to provide input for these fields when you use ENTRO for data entry. This can be useful if you want to hide or protect certain fields. Field names that do not appear in the @ENTRO phrase are protected from user modification, because there will be no way to view them with the ENTRO processor. See Note below for more information on data security in ENTRO.

Use either ENTRO or the Editor to modify the @ENTRO phrase. If an @ phrase already exists in the dictionary when ENTRO is first invoked, an @ENTRO phrase will not be created. Instead, the @ phrase will be used just like an @ENTRO phrase.

Note

To restrict users from adding any data to a file with the ENTRO processor, edit the @ENTRO phrase (or create one yourself from scratch) and initialize the second field to null. The second field normally contains the list of data descriptors for which field values are to be supplied by the user. If no names are listed, ENTRO cannot display any field names, and therefore cannot prompt for input.

Sample INFORM Phrases: The STUDENTS file dictionary contains both an @ phrase and an @LPTR phrase. The @ phrase was created prior to adding any data to the data file. Because an @ phrase already existed in the dictionary, ENTRO did not create an @ENTRO phrase. Example 2-29 shows the contents of the @ and @LPTR records respectively.

```
:LIST DICT STUDENTS "@" "@LPTR"

LIST DICT STUDENTS "@" "@LPTR" 16:51:38  08-14-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@              PH   FNAME LNAME
                    COURSES.GRADES
@LPTR          PH   FNAME LNAME
                    GRAD.YR

2 records listed.
```

Example 2-29. Special @ Phrases

DATA FILES

As already mentioned, a data file contains all the information to be stored for a particular application. In order to put information in a file, you must first describe the logical structure of a typical file record. Since each file record is stored as a variable-length ASCII string, you really don't have to worry about defining a maximum or minimum size for each record. This means that the fields in the record don't have to be a fixed length either.

Estimating File Size

When you create a file using the CREATE.FILE verb, you have to estimate how big you think the average record will be. This helps you estimate a modulus for the file.

Note

> The modulus specifies the number of groups in which file records will be stored. The most efficient file organization can be achieved by keeping the group size as close to 1900 bytes as possible. To determine the modulus for the file, you must know the approximate number of records the file will contain and the average record size.

The modulus can be changed if you discover that you really have a lot more data to store than you originally anticipated. The reverse case is also true, that is, you can decrease the modulus size if it turns out that you have fewer data records than you originally anticipated. Use the results obtained by using the PERFORM HASH.HELP, HASH.TEST, and GROUP.STAT commands in making decisions of this type. For more information on these commands and on hash-encoded addressing, see the PERFORM Reference Guide. If you think your file may not be structured as efficiently as it could be and you're not sure what to do about it, see your System Administrator for help.

Creating a Data File

The steps involved in setting up a data file are:

1. Decide what data elements (fields) the file should contain.

2. Choose a field to serve as a unique record identifier for each record.

3. Choose a file type based on the data characteristics of this record identifier (called record ID).

4. Estimate the size of the file and pick a suitable modulus.

5. Use CREATE.FILE in the form:

   CREATE.FILE filename [type] [modulus] [description]

   to create both the data and dictionary files. The type, modulus, and description arguments do not have to be supplied on the command line, as PERFORM will prompt for them if they're not supplied.

6. Use ENTRO to describe the file's logical structure:

   ENTRO DICT filename

7. Once the dictionary file has been defined, use ENTRO to add data to the data file:

   ENTRO filename [field.name] [field.name]...

ENTRO isn't the only method available for adding data to a file. The INFORMATION Editor is another option for data entry, providing you're familiar with the structure of the data file record you're dealing with. An INFO/BASIC program can be designed to do the same thing. See the INFO/BASIC Reference Guide for details.

Adding Data With ENTRO

Invoke ENTRO by typing:

ENTRO filename

where filename is the name of the file to which you want data added. If no field names are supplied, the @ENTRO phrase, if it exists, will be used instead. ENTRO will then prompt you for the field information required. A data entry screen will appear on the terminal with the names of the fields as they exist in the dictionary.

Data Entry Screens

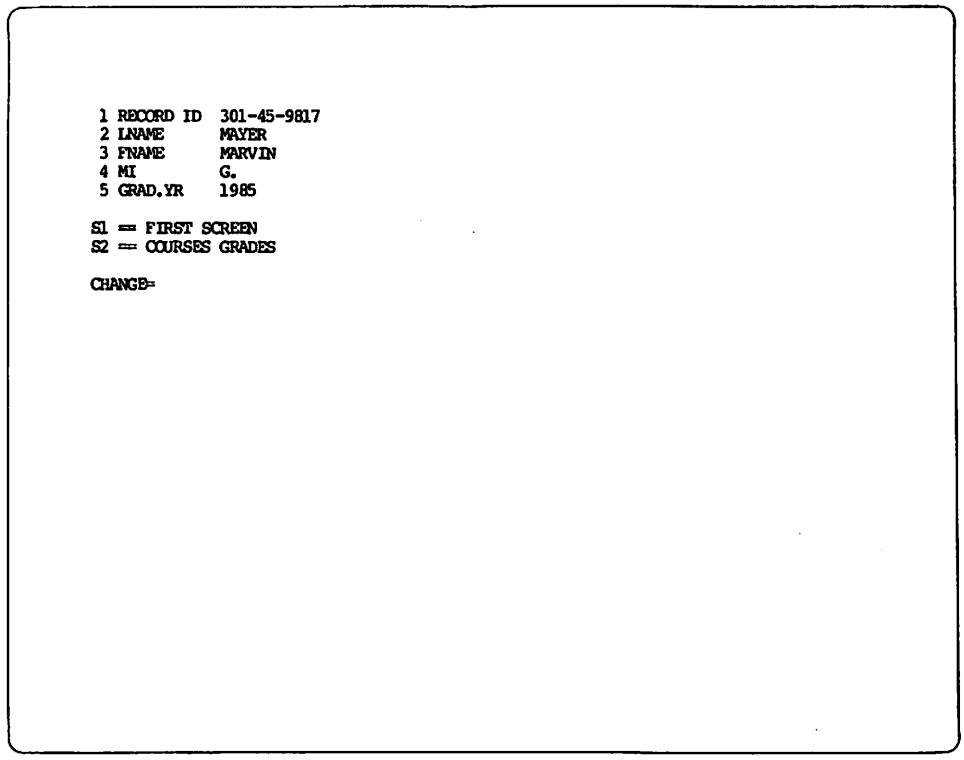
When you enter records into the data file, ENTRO uses a default screen display to prompt for each field defined for the file. There are two types of screens. All single-valued fields are prompted for in the same screen, which is always called SCREEN 1. However, if you have multivalued fields in a record, ENTRO automatically creates additional screens, one for each multivalued field, beginning with SCREEN 2. These special multivalue screens allow you to add more than one value for a particular field. If the fields are in an association, their names will appear togther in a screen so that the data can be handled properly. See Section 3 under DATA ENTRY AND MODIFICATION, for more information.

Single Value Screens: For example, when adding data to the STUDENTS file, the ENTRO screen shown in Example 2-30 comes up.

```
:ENTRO STUDENTS
STUDENTS  ENTRO.1  14:30:03 25 JUN 1981

SOC-SEC-NO=301-45-9817
New record
LAST NAME=MAYER
FIRST NAME=MARVIN
MID. INIT.=G.
GRAD YEAR=1985
```

Example 2-30. ENTRO Single Value Data Entry Screen

A Chance to Change: When you've finished adding all the data for a particular record, ENTRO then repaints the screen, allowing you to change any of the field values that you have entered. If the file contains only single valued fields, that is, data descriptors with an S in the S/M field, each record you add will be listed in its entirety so you can see the just added values. If the file contains multivalued fields, the process is a bit different, as explained below. Example 2-31 shows the Screen 1 change prompts.



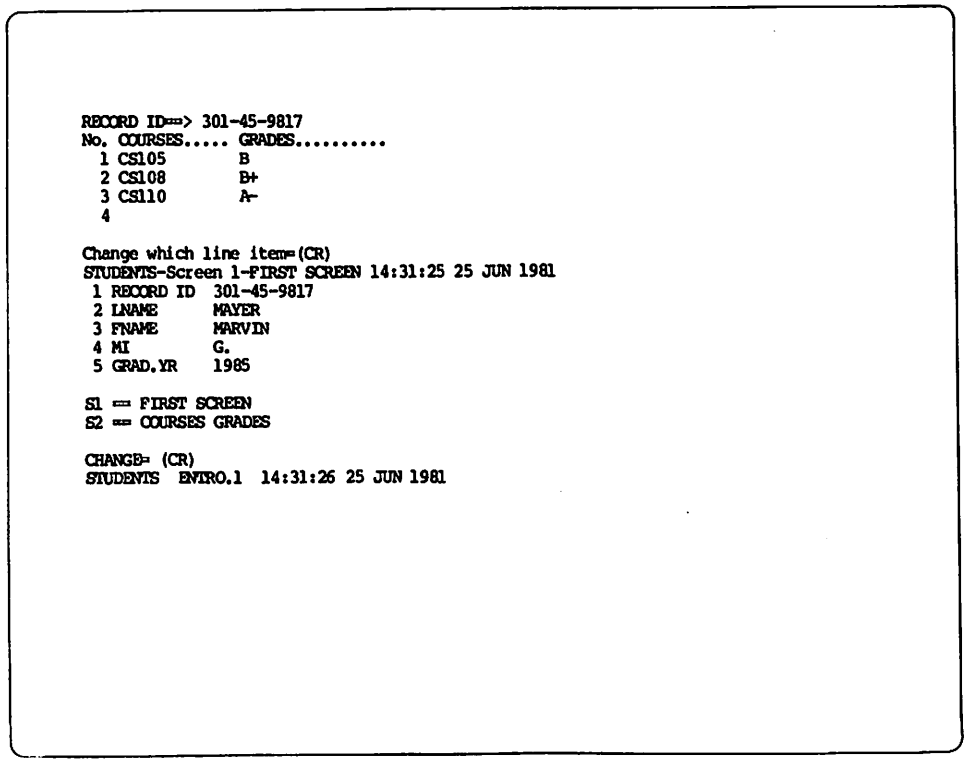
Example 2-31. ENTRO's Screen 1 Change Prompts

Multivalue Screens: Because COURSES and GRADES are multivalued fields, ENTRO creates a separate screen to prompt for values for these fields. These fields are linked in an association (COURSES.GRADES) so ENTRO prompts for them in the same screen. As soon as you finish adding a new set of values for associated descriptors, those values are repainted on the screen and you have a chance to change them or add new ones. Example 2-32 shows how ENTRO prompts for information for the COURSES and GRADES fields. In this example, values were entered for two courses, CS105 and CS108.

```
STUDENTS-Screen 2-COURSES GRADES 14:31:00 25 JUN 1981
RECORD ID==> 301-45-9817
No. COURSES..... GRADES.....
  1
COURSE NUMBER=CS105
GRADE FOR COURSE=B
STUDENTS-Screen 2-COURSES GRADES 14:31:05 25 JUN 1981
RECORD ID==> 301-45-9817
No. COURSES..... GRADES.....
  1 CS105        B
  2
COURSE NUMBER=CS108
GRADE FOR COURSE=B+
```

Example 2-32. ENTRO's Multivalue Data Entry Screen

After adding a third value, ENTRO prompts for a fourth, but it was decided not to add any more at this time. ENTRO then asks if any of the values that were just added should be changed. If you don't want to change anything, enter a null line (CR) and ENTRO displays all the single valued fields for the current record plus the S1 and S2 indicators. The S2 = COURSES GRADES line indicates that there are two multivalued fields linked in an association because they appear in the same ENTRO screen. To change anything in the second screen, type S2 in response to the CHANGE= prompt. Example 2-33 shows the sequence of events just described.



Example 2-33. Changing Multivalued Fields

Changing Data in Associated Fields: As you are adding values to a multivalued field, ENTRO gives you a chance to change them after each new set is added. Each of the subvalues is numbered so you can specify which ones you want to change.

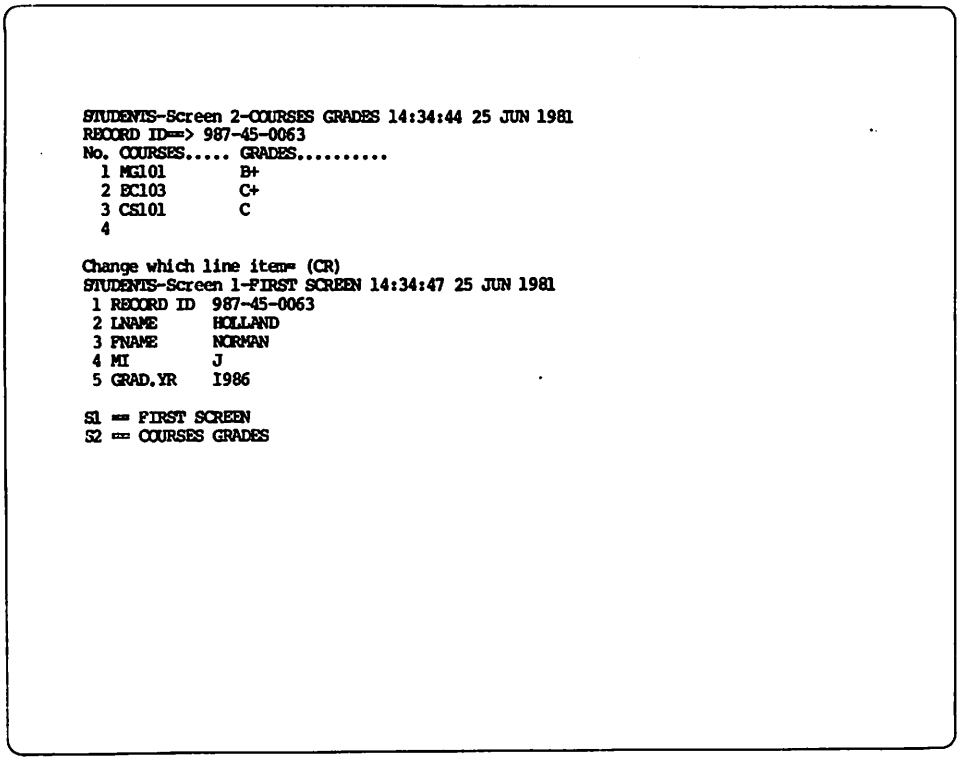
Examples 2-34 and 2-35 show how to change values in a multivalued field as data is being added to the file.

```
RECORD ID==> 987-45-0063
No. COURSES..... GRADES.....
  1 MG101        B+
  2 EC103        C+
  3 CS101        C+
  4
COURSE NUMBER= (CR)
STUDENTS-Screen 2-COURSES GRADES 14:34:34 25 JUN 1981
RECORD ID==> 987-45-0063
No. COURSES..... GRADES.....
  1 MG101        B+
  2 EC103        C+
  3 CS101        C+
  4

Change which line item=3
STUDENTS-Screen 2-COURSES GRADES 14:34:38 25 JUN 1981
RECORD ID==>>987-45-0063  Line==>> 3
 6 COURSES     CS101
 7 GRADES      C+
CHANGE=7
GRADE FOR COURSE=C
CHANGE= (CR)
```

Example 2-34. Changing Values in Associated Fields (Part 1)

Three courses had already been added to this student's record. ENTRO prompted for a fourth set of values, but only the third set was to be changed, so the RETURN key was hit (indicated by (CR)) in response to the COURSE NUMBER= prompt. ENTRO then asks which set should be changed. The values for the third set are then displayed.

After all the appropriate changes are made and no more entries are to be added for this record, ENTRO repaints the screen, giving us another chance to change the values just added, as Example 2-35 shows.



Example 2-35. Changing Values in Associated Fields (Part 2)

If you don't want to make further changes at this point, ENTRO gives you the opportunity to add another data file record, or to exit the data entry sequence entirely.

Modifying a Data File

If you want to change a field that was entered incorrectly or you just want to add some new records, use the same form of the ENTRO command shown previously. Simply enter the record ID that corresponds to the record you want to change and ENTRO will paint that record on the screen. ENTRO then asks in which screen is the field you want to change. These screens are just like the ones ENTRO uses to allow you to make changes during data entry.

In Example 2-36, ENTRO is invoked to change a field value in the record describing the student Adelle Jarvis. To call up this record, you have to supply ENTRO with the appropriate social security number in response to the SOC-SEC-NO= prompt. To change the value in Field 4 of Screen 1 (S1), enter the number 4 in response to the CHANGE= prompt.

```
SOC-SEC-NO=131-43-5670
STUDENTS-Screen 1-FIRST SCREEN 14:33:08 25 JUN 1981
 1 RECORD ID  131-43-5670
 2 LNAME      JARVIS
 3 FNAME      ADELLE
 4 MI
 5 GRAD.YR    1983

S1 = FIRST SCREEN
S2 = COURSES GRADES

CHANGE=4
MID. INIT.=G
CHANGE= (CR)
STUDENTS  ENTRO.1  14:33:22 25 JUN 1981


SOC-SEC-NO=131-43-5670
STUDENTS-Screen 1-FIRST SCREEN 14:33:33 25 JUN 1981
 1 RECORD ID  131-43-5670
 2 LNAME      JARVIS
 3 FNAME      ADELLE
 4 MI         G
 5 GRAD.YR    1983

S1 = FIRST SCREEN
S2 = COURSES GRADES
```

Example 2-36. Changing Data in Screen 1

Examples 2-37 and 2-38 show how to access values in Screen 2. To look at field values in Screen 2 (S2), specify S2 in response to the CHANGE= prompt, as shown in Example 2-37.

```
:ENTRO STUDENTS
STUDENTS  ENTRO.1  15:58:59 18 AUG 1981

SOC-SEC-NO=131-43-5670
STUDENTS-Screen 1-FIRST SCREEN 15:59:12 18 AUG 1981
 1 RECORD ID  131-43-5670
 2 FNAME      ADELLE
 3 LNAME      JARVIS

S1 = FIRST SCREEN
S2 = COURSES GRADES

CHANGE=S2
STUDENTS-Screen 2-COURSES GRADES 15:59:15 18 AUG 1981
RECORD ID==> 131-43-5670
No. COURSES..... GRADES.....
  1 CS101        B
  2 CS301        B
  3 CS579        A-
  4 CS673        D
  5 CS673        B+
  6

Change which line item=2
```

Example 2-37. Changing Data in Screen 2 (Part 1)

After you indicate which of the values in the second screen is to be changed, in this case the GRADES value for the course CS301, ENTRO repaints the values. Again, you have a chance to change things in the second screen or go back to the intial prompt screen. Example 2-38 illustrates the sequence just described.

```
STUDENTS-Screen 2-COURSES GRADES 15:59:44 18 AUG 1981
RECORD ID==>>131-43-5670  Line==>> 2
 4 COURSES     CS301
 5 GRADES      B
CHANGE=5
COURSE GRADE=A
CHANGE=(CR)
STUDENTS-Screen 2-COURSES GRADES 15:59:53 18 AUG 1981
RECORD ID==> 131-43-5670
No. COURSES..... GRADES......
  1 CS101        B
  2 CS301        A
  3 CS579        A-
  4 CS673        D
  5 CS673        B+
  6

Change which line item=(CR)
STUDENTS-Screen 1-FIRST SCREEN 15:59:57 18 AUG 1981
 1 RECORD ID  131-43-5670
 2 FNAME      ADELLE
 3 LNAME      JARVIS

S1 = FIRST SCREEN
S2 = COURSES GRADES

CHANGE=(CR)
```

Example 2-38. Changing Data in Screen 2 (Part 2)

More on Changing Data

There are many ways to change data in a file using the ENTRO processor. These are discussed in Section 3 under DATA ENTRY/MODIFICATION.

SECTION 3

INFORM VERBS

INTRODUCTION

There are nine major verbs in the INFORM system. With them, you can add, delete, modify, display, sort, count, and sum file information. A variety of INFORM keywords expand the functionality of these verbs, making possible things like report formatting, displays with columns and headings, reverse sorting, breakpoints and subtotals in reports, and so forth.

Verbs that are part of INFORM require the use of a file dictionary in order to operate while PERFORM verbs, also called commands, in general, do not. INFORM verbs handle user queries for particular file information based on some field descriptor value or on a condition that involves several fields and values. Such file interrogations, or searches, require the file dictionary, since the dictionary indicates the relative location of each field in the associated data file, as well as the output format specification for each field.

What's in This Section

This section describes the formats and general uses of all the INFORM verbs. Each of these verbs can be used with any number of available INFORM keywords. Where possible, a list of these keywords is given, or a reference to the applicable class of keywords is given so you can find them in Section 4. All keywords referred to in this section are covered in Section 4.

The topics covered in this section are:

- The INFORM Verbs
- Data Entry and Modification
- Data Retrieval and Display
- Select List Operations
- Arithmetic Operations
- Magnetic Tape Operations

THE INFORM VERBS

The nine INFORM verbs and the four PERFORM verbs most often used in conjunction with them are listed in Table 3-1.

Table 3-1
INFORM Verbs and Associated PERFORM Verbs

Verb	What It Does
COUNT	Counts the number of records in a specified file.
DELETE.LIST	(PERFORM verb) Deletes a saved select list.
ENTRO	Adds, changes, or deletes records in a dictionary file or in a data file.
FORM.LIST	(PERFORM verb) Forms a select list from a user-built list of record IDs.
GET.LIST	(PERFORM verb) Retrieves and makes active a saved select list.
LIST	Displays the contents of a dictionary or data file.
SAVE.LIST	(PERFORM verb) Names and saves a select list of record IDs for repeated use.
SELECT	Creates a select list of certain records from a specified file to be used for further processing.
SSELECT	Same as SELECT but sorts records in a list by record ID.
SORT	Same as LIST but the records are sorted by record ID.
SUM	Adds up numeric values in specified fields.
T.DUMP	Copies disk records to tape with optional blocking, sorting, and selection criteria.
T.LOAD	Copies records from tape to disk. Only tapes saved with T.DUMP can be used with T.LOAD.

In Table 3-2, the INFORM verbs are presented in a functional, rather than alphabetical order.

Table 3-2
INFORM Verbs Listed By Function

Function	Verb
Data Entry and Modification	ENTRO
Data Retrieval and Display	LIST and SORT
Using Select Lists	SELECT, SSELECT, SAVE.LIST, GET.LIST, FORM.LIST, DELETE.LIST
Arithmetic Operations	COUNT and SUM
Magnetic Tape Operations	T.DUMP and T.LOAD

DATA ENTRY AND MODIFICATION

The ENTRO processor is a multipurpose tool for adding, deleting, examining and changing data in both dictionary and data files. Earlier in Section 2, there was a discussion about how ENTRO could be used to create the contents of a dictionary file. The basic prompting sequences and useful features of ENTRO were also introduced. However, ENTRO has many other conventions available for looking at and modifying file information. These conventions and how to use them are all covered in this section.

ENTRO Syntax

There are two versions of the ENTRO processor: ENTRO (also called ENTER), and ENTROC, which does cursor control and screen formatting on ADDS (Regents) terminals. Both versions prompt for record input or modification, and they check that the data entered is valid for the conversion specification indicated for that field. The ENTROC version indicates the length of each field with an arrow, as in this display:

```
COURSE.NO=      <=
```

Since the COURSE.NO field is only five characters in length, the arrow is positioned five spaces over from the field name prompt. This is only a guideline for input, as no error message is displayed if the data entered exceeds the indicated length. The ENTROC feature works on the ADDS terminals and others which have compatible cursor addressing capabilities.

To invoke either form of ENTRO, type:

```
{ENTRO }
{ENTER } [DICT] filename [field.name]... [phrase.name]...
{ENTROC}
```

The field.name parameter can be supplied when you want to modify or add things to selected fields in a record. A phrase.name can be supplied in place of field names if the phrase contains the names of fields you want to work on. Field names or phrase names should not be supplied when the DICT keyword is used; they are meaningful only when applied to a data file. Only D-descriptors are recognized; all others are ignored.

Working on the Dictionary: Use the DICT keyword to add or modify dictionary file records. ENTRO's dictionary file prompting sequence is always the same. It asks first for a record ID value, or field name. It then asks for a descriptor type and optional description for this field. The prompts are the same whether you create a data descriptor, I-descriptor, or a phrase, but the meaning of the data varies, as described earlier in Section 2.

Working on the Data File: If the DICT keyword is omitted, ENTRO will prompt for changes or additions to the data file. When field or phrase names are included in the ENTRO sentence, ENTRO will only prompt for the named fields. If a select list is active, ENTRO brings up the records contained in the select list. (A select list is a set of record ID values formed by the SELECT verb, described later in this section.)

When you have finished looking at the indicated records, ENTRO will then ask you for another record ID value. If you haven't changed the name of the record ID descriptor, ENTRO prompts for a record ID value by displaying whatever appears in the Display Name Field (Field 4) of the @ID record. Unless you have changed this field, it contains the name of the file by default. If the record ID you supply doesn't exist in the file, ENTRO assumes you want to add a new record and proceeds accordingly. If a record is found with this key value, it is brought up for inspection and modification.

Adding New Records

Every time a record ID value is entered, ENTRO checks to see if it already exists in the file. Remember that duplicate keys are not allowed. If the value does not exist, ENTRO displays the NEW RECORD message and then prompts for the field information for this record using the @ENTRO phrase (if one exists) in the file dictionary. If no @ENTRO phrase exists in the dictionary, ENTRO looks for a user-created @ phrase to use instead. When neither the @ phrase nor the @ENTRO phrase exists, ENTRO creates an @ENTRO phrase using the names of the data descriptors currently in the dictionary. This means that if you add new data descriptors or remove existing ones after an @ENTRO phrase is created, you will have to update the @ENTRO phrase to reflect the current state of the dictionary. More on this later in the section.

Note

> The @ENTRO phrase can be used to restrict user access to certain fields in a record by making it impossible for these fields to be seen via ENTRO. Simply do not include the names of any fields you want to protect in the @ENTRO or @ phrase. Any fields not named therein will not be displayed for entry or inspection.

To prompt for record information, ENTRO uses the data descriptor names contained in the @ phrase, or in the @ENTRO phrase, or in the field or phrase name specified in the ENTRO sentence itself. Once you have responded to all the fields for a particular record, ENTRO then repaints the screen, giving you a chance to change the values you supplied for the new record. There are a number of things you can do when ENTRO prompts for changes. Possible responses are included under ENTRO Conventions below. If you don't want to make any changes, ENTRO files the record and then prompts for another record ID. If you don't want to add or modify any records, type QUIT or simply hit RETURN.

ENTRO and Select Lists

If a select list is active when ENTRO is invoked, the appropriate set of records will be brought up, one at a time, for modification. If the select list contains record IDs which do not exist in the file, ENTRO will prompt for the information needed to add a new record. To get to the next record in the set, simply hit a carriage return in response to the CHANGE= prompt and the next record will come up automatically. When the select list is exhausted, ENTRO then prompts for a record ID value as above. To return to ENTRO's normal prompting mode while a select list is active, simply type QUIT in response to the CHANGE= prompt, and ENTRO will ask for a record ID value.

Note

If QUIT is typed in response to the CHANGE prompt while a select list is active, then the current record is not modified. When you type QUIT in response to the CHANGE= prompt, when there is no active select list, ENTRO will display a message indicating that there is no select list active. Then ENTRO will redisplay the CHANGE= prompt.

Multivalued Fields: ENTRO handles multivalued fields by isolating each one on a separate screen. ENTRO prompts for all single-valued fields using Screen 1 which simply displays the name of the field, accepts a single value, and then brings up the next field. The version of Screen 1 used for data modification, numbers and lists all the single-valued fields in columnar format, displaying their current values after them. ENTRO prompts for multivalued fields with a special second screen (S2) format that allows more than one value to be added to the field. This multivalue screen also displays all the values you've just added as you go along. This gives you a chance to change things as you proceed. A separate second screen format is created for each multivalued field in the file.

ENTRO and Associations: If a multivalued field is a member of an association, then the entire association is displayed in the row format, with the field names across the top of the screen. Any member of the association not specified in the field name list will only be displayed, and cannot be changed. If new values for the association are entered, these display-only fields will be automatically created as null values. Each row-format screen will accept an indefinite number of values, paging automatically. When displaying existing values, the process will display one screenful at a time, pausing on the CHANGE WHICH LINE NUMBER prompt. When a blank line (a single carriage return) is entered as a response to this prompt, either the next multivalued field will be displayed, or the first screen will be redisplayed.

ENTRO Conventions

ENTRO's many conventions make data modification flexible but simple. At nearly every point, you can cancel everything you've done, start over, back up to the previous prompt, repaint the screen, get more information on what is expected as a response, or simply skip on to the next step. The ENTRO conventions you'll need to do these things can be divided into these topics:

- General Responses to ENTRO Prompts
- Responses to Initial (Record ID) Prompt
- Responses to the CHANGE= Prompt
- Making Changes
- Changing Multivalued Fields

General Responses to ENTRO Prompts

ENTRO has a number of data entry and data modification screens which it uses to prompt for information and directions. The prompts used in these screens accept a variety of user responses. Some responses are accepted by all of ENTRO's prompts and always do the same thing, regardless of where they are used. These generally accepted responses to ENTRO prompts are described in Table 3-3.

Table 3-3
Valid User Responses for all ENTRO Prompts

Response	Action
?	Displays a brief explanation of the field, if user provided one during dictionary setup.
^	Backs up and repeats the previous prompt.
TOP	Bails out on the current transaction, leaving record unchanged.
.	In input mode, skips to the next required prompt.
^^	Repaints the screen and re-prompts.
??	Displays an extended explanation of prompt, if one is provided.

Responses to Initial (Record ID) Prompt

The first prompt that ENTRO displays when invoked (if not using a select list) is the Record ID prompt. The name that appears in the Display Name field of the @ID record is used as a prompt. The legal responses to this prompt are listed in Table 3-4.

Table 3-4
User Responses to Record ID Prompt

Response	Action
record-ID-value	If the value you enter already exists in the file, the record is brought up to be modified. If the value is a new one, ENTRO then goes into input mode and prompts for the appropriate field information.
(CR)	Hitting RETURN or NEWLINE exits you from ENTRO at this point, unless you used the NEXT.AVAILABLE keyword in your INFORM sentence. See Section 4 for details.
QUIT	Exits you from ENTRO. Same as (CR).
^^	Displays all the fields for which ENTRO would prompt if you were adding a new record. The initial prompt is then repainted on the screen.

Responses to the CHANGE= Prompt

If you enter an existing record ID value in response to the initial ENTRO prompt, the record will be painted on the screen and the CHANGE= prompt will appear. Each field in the record is numbered so that you need only supply that number in order to modify the field value. You have several options at this point. They are listed in Table 3-5.

Table 3-5
User Responses to CHANGE= Prompt

Response	Action
prompt.number	Displays the field name that corresponds to this ENTRO-provided number. This allows you to make changes to this field. (See below.)
(CR)	Indicates that you do not wish to make any changes. Causes ENTRO to prompt for another record ID.
DELETE	Deletes the current record.
Snn	Brings up indicated screen number if it is defined and listed on the first screen (S2 = LINE ITEMS).
QUIT	Used only if processing a select list. ENTRO prompts for another record ID and current record is not filed. Changes are ignored.

Note

The responses listed above under General Responses to ENTRO Prompts may also be used in replying to the CHANGE= prompt.

Changing Field Values

When you respond to the CHANGE= prompt with the number of the field you want to change (prompt.number), there are several options:

- Use the ENTRO C command to change any portion of the existing field value. The C command has the format:

  C/old.string/new.string/

  ENTRO will change old.string to new.string within the field.

- Retype the entire field value with the new value you want in the field. (Simply enter the new value.)

- Hit a carriage return (CR) and leave the field empty.

In addition to these responses, any of the general responses listed above will also work.

Changing Multivalued Fields

If the file being modified has multivalued fields, ENTRO will indicate that there is a separate screen for each of these fields when it displays any record you want to modify. For example, the STUDENTS file has two multivalued fields in an association called COURSES.GRADES. Since the @ phrase (which ENTRO automatically uses when no other fields are specified to be modified) contains this association name, the following is painted on the screen:

```
:ENTRO STUDENTS

SOC-SEC-NO=131-43-5670
 1 RECORD ID   131-43-5670
 2 FNAME       ADELLE
 3 LNAME       JARVIS

S1 == FIRST SCREEN
S2 == COURSES GRADES

CHANGE=S2
RECORD ID=> 131-43-5670
No. COURSES..... GRADES.....
  1 CS101        B
  2 CS579        A-
  3
```

Example 3-1. ENTRO's Multivalue Screen

To change anything in the second screen, type S2 in response to the first CHANGE= prompt. The names of the fields in the association are then displayed and numbered so you can indicate which ones you want to examine and/or modify.

Responses to Line Item Prompt: The next ENTRO prompt in the Screen 2 sequence is:

Change which line item=

ENTRO wants to know which of the items in association are to be modified. The responses listed in Table 3-6 are accepted as valid responses to this prompt.

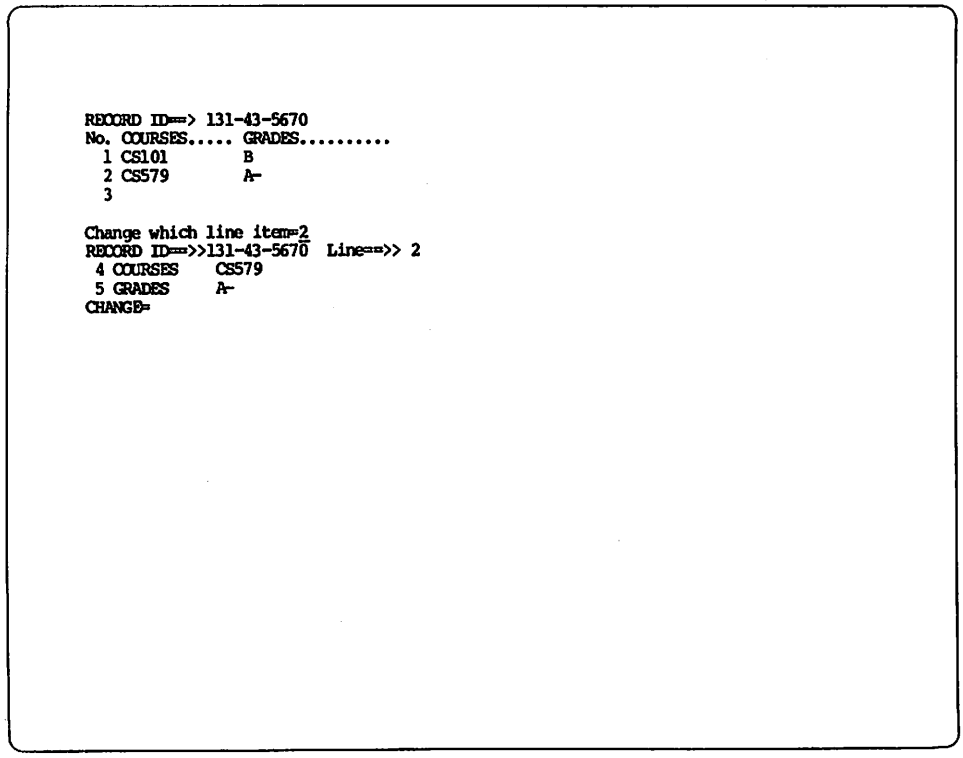
Table 3-6
User Responses to Line Item Prompt

Response	Action
line number	Displays the contents of the field with this ENTRO-assigned number. Field values are displayed in vertical format and are numbered so you can specify exactly which values are to be altered.
(CR)	Displays next page of information if the record contains more values than can fit on a single screen; or Displays the next horizontal screen if the user is in input (data entry) mode and if there are more screens defined for the file; or Displays the first screen again, if neither of the above conditions are true.

Note

In addition, any of the responses listed under General Responses to ENTRO Prompts are also acceptable.

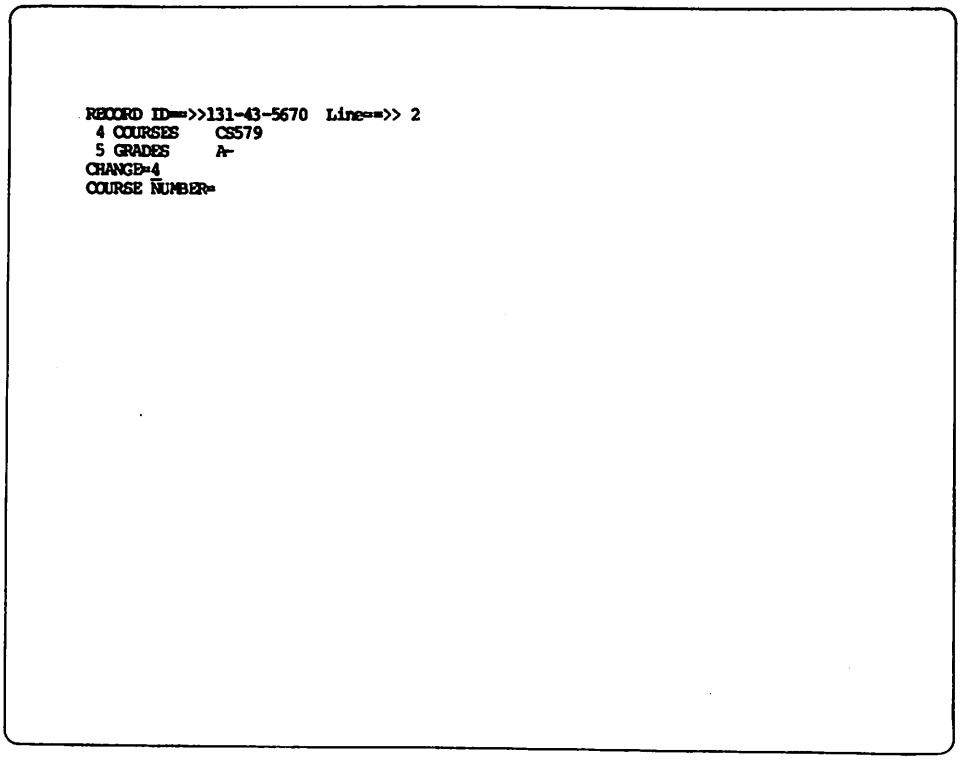
Continuing with the above example, if you wanted to change the field information in line number 2, (CS579), you would type a 2 in response to the last prompt, as shown in Example 3-2.



Example 3-2. ENTRO's Prompts for Multivalued Fields

Then you would indicate which of these values in the association are to be changed. This example continues below.

Responses to Screen 2 CHANGE= Prompt: If you enter the number of a field to be modified, the values that exist in this field will be displayed, along with ENTRO-supplied numbers, so you can see which fields you want to change. All of the fields which ENTRO can display for modification are numbered consecutively, beginning with the fields in Screen 1. Thus, in this example, the COURSES and GRADES fields are numbered 4 and 5 respectively. In the same way, SOC-SEC-NO, FNAME, and LNAME are numbered 1, 2, and 3 respectively. To indicate which field should be changed, enter the proper number in response to the CHANGE= prompt. This is shown in Example 3-3.



Example 3-3. Changing Multivalued Field Items

The display name which corresponds to the numbered field is then displayed, and ENTRO waits for you to do something with this value.

Changing Multivalued Fields: After you specify the line number of the field value you want changed, ENTRO displays the field name and waits for you to modify the value. You can retype the value, use the special ENTRO C command to change a part of it, you can delete it, insert a new value above it, or copy the previous entry. To accomplish these things, use the conventions listed under Responses to the CHANGE= Prompt above, or use any of the special ones listed in Table 3-7 that work only for multivalued fields.

Table 3-7
Responses for Changing Multivalued Fields

Response	Action
##	Deletes the line you asked to modify.
>	Indicates that you want to insert a new line above the current one. ENTRO then displays an explanatory message and asks for the new line.
>#nn	Inserts a copy of line number nn, as listed in the current display, above the current line, which is the one currently being modified.
"	Copies the entry from the line above the current one, if one exists.

Examples 3-4 through 3-6 show how to use some of ENTRO's conventions for multivalued fields. They also illustrate how to get out of the ENTRO prompting sequence in multivalue screens.

```
:ENTRO STUDENTS

SOC-SEC-NO=131-43-5670
 1 RECORD ID  131-43-5670
 2 FNAME      ADELLE
 3 LNAME      JARVIS

S1 == FIRST SCREEN
S2 == COURSES GRADES

CHANGE=S2
RECORD ID==> 131-43-5670
No. COURSES..... GRADES.....
  1 CS101        B
  2 CS579        A-
  3

Change which line item=2
RECORD ID==>>131-43-5670  Line==>> 2
 4 COURSES    CS579
 5 GRADES     A-
CHANGE=4
COURSE NUMBER=>
Inserting the following information at line 2.
COURSE NUMBER=CS301
COURSE GRADE=B
```

Example 3-4. Changing Screen 2 Values (Part 1)

```
:ENTRO STUDENTS
STUDENTS  ENTRO.1  11:38:41 09 SEP 1981


SOC-SEC-NO=131-43-5670
STUDENTS-Screen 1-FIRST SCREEN 11:38:47 09 SEP 1981
 1 RECORD ID  131-43-5670
 2 FNAME      ADELLE
 3 LNAME      JARVIS

S1 == FIRST SCREEN
S2 == COURSES GRADES

CHANGE=S2
STUDENTS-Screen 2-COURSES GRADES 11:38:50 09 SEP 1981
RECORD ID==> 131-43-5670
No. COURSES..... GRADES......
  1 CS101        B
  2 CS301        A
  3 CS579        A-
  4 CS673        D
  5

Change which line item=5
COURSE NUMBER= >#4
STUDENTS-Screen 2-COURSES GRADES 11:39:32 09 SEP 1981
RECORD ID==>>131-43-5670  Line==>> 5
 4 COURSES    CS673
 5 GRADES     D
COURSE NUMBER= (CR)
```

Example 3-5. Changing Screen 2 Values (Part 2)

```
STUDENTS-Screen 2-COURSES GRADES 11:39:41 09 SEP 1981
RECORD ID==> 131-43-5670
No. COURSES..... GRADES......
  1 CS101        B
  2 CS301        A
  3 CS579        A-
  4 CS673        D
  5 CS673        D
  6

Change which line item=5
STUDENTS-Screen 2-COURSES GRADES 11:39:43 09 SEP 1981
RECORD ID==>>131-43-5670  Line==>> 5
 4 COURSES    CS673
 5 GRADES     D
CHANGE=5
COURSE GRADE=B+
CHANGE= (CR)
STUDENTS-Screen 2-COURSES GRADES 11:40:04 09 SEP 1981
RECORD ID==> 131-43-5670
No. COURSES..... GRADES......
  1 CS101        B
  2 CS301        A
  3 CS579        A-
  4 CS673        D
  5 CS673        B+
  6

Change which line item= (CR)
STUDENTS-Screen 1-FIRST SCREEN 11:40:16 09 SEP 1981
 1 RECORD ID  131-43-5670
 2 FNAME      ADELLE
 3 LNAME      JARVIS

S1 == FIRST SCREEN
S2 == COURSES GRADES

CHANGE=(CR)
```

Example 3-6. Changing Screen 2 Values (Part 3)

In Example 3-4, an entry for CS301 was inserted above the entry for CS579, using the > convention. In Example 3-5, a copy of the CS673 entry was made, using the >## convention. The student took the same course again, not having done well the first time. The entry was copied into line number 5. The GRADES entry was then changed in Example 3-6 to reflect the new grade for the course.

Note

If you use ENTRO to delete a value from a multivalued field in an association, the other fields associated with it in the current record will be deleted as well.

ENTRO Keywords

The INFORM keywords designed specifically for use with the ENTRO processor are:

- VERIFILE (VERIFY) checks that all input for a certain field exists as a record ID in another file.
- VERIFIELD is used with VERIFILE to display the contents of any field in the VERIFILE file.
- NEXT.AVAILABLE tells ENTRO to generate new sequential numeric record IDs based on the contents of a special dictionary record which is created automatically by ENTRO the first time this keyword is used.
- LIKE (MATCHES, MATCHING) restricts input values for certain fields to those matching a specified pattern.

These keywords are documented in Section 4.

DATA RETRIEVAL AND DISPLAY

The most versatile of the INFORM verbs is LIST. It can do everything from a simple search on a record ID value to a complex report, depending on the options you use. LIST performs both a search and a display, making it possible for you to accomplish two separate actions with a single verb or sentence. LIST can also sort output records by one or more fields, in ascending or descending order. The SORT verb has the same functionality as LIST, except that output is automatically sorted by record ID value.

The LIST Verb Format

The LIST verb format looks complex, but it doesn't have to be. It simply provides a lot of room to get fancy. The basic format, showing all the required and optional parts, is:

LIST [DICT] filename [field names] [selection criteria] [sort criteria] [output options] [record IDs]

The only required parameter in a LIST sentence is the filename. All the rest are optional, depending on what you want to find in the file or produce as output. The order in which these parameters can appear in the sentence is not rigid, except that the filename should always be specified immediately after the verb. Field names can come before or after the selection criteria and other options. Any record ID values are generally placed at the end of the sentence for clarity. When listing fields from a dictionary file (using the DICT option), enclose any record ID values in quotes. (In dictionary files, record ID values are descriptor names.)

Field Names: Use field names to indicate the fields you actually want INFORM to display. In a sense, these field names are really "display names" or "output names." If no field names are specified, LIST simply displays the record ID values of the records it found as a result of the search. Should the dictionary contain an @ phrase, the field names listed there will be displayed if no field names are specified. More information on this is provided in LIST Defaults below.

Note

INFORM always displays the record ID field when it executes a LIST or SORT sentence unless the ID.SUP keyword, which suppresses the record ID display, is included. ID.SUP keyword is discussed in Section 4.

Selection Criteria: If you only want certain records from a file to be displayed, a search expression which states selection criteria must be included in the LIST sentence. Selection criteria include things like: "LNAME EQ KATZ or AGE GT 30." More on this can be found under Selection Expressions later in this section.

Sort Criteria: To sort the output of the LIST operation on one or more fields, use the "BY" or "BY.DSND" keywords followed by the field name on which to sort. If no sort criteria are specified, the records will be listed out in sequential order according to how they're stored.

Options: This is a general term indicating other INFORM keyword options which can be used to format output, suppress the record ID display, send the display to the line printer, and so forth. See Section 4 for details on all the available options.

Record IDs: If you know the record ID values of the records you want displayed, simply include them on the LIST verb line. Although it isn't necessary, it is a good idea to put these key values in quotes to make the sentence easier to read. In that way a reader knows exactly which items are field names and which are record ID values. For example:

```
LIST COSTS COURSE.NAME PRICE "CS101" "CS673"
```

The records corresponding to the key values of CS101 and CS673 will be found and the COURSE.NO, COURSE.NAME, and PRICE fields will be displayed. (Remember that the record ID values, in this case, those values in the COURSE.NO field, are always displayed unless you use the ID.SUP keyword.)

LIST Verb Defaults

If no options are supplied in a LIST sentence, INFORM does several things for you automatically. Exactly what is done depends on what special phrases exist in your dictionary file.

If an @ Phrase Exists: If the dictionary for the indicated file contains an @ phrase, the fields listed in this phrase will be displayed. In the COSTS file, for example, the @ phrase is defined as shown in Example 3-7.

```
:LIST DICT COSTS "@"

LIST DICT COSTS "@" 15:36:55  10-21-81  PAGE      1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@               PH   COURSE.NO
                     COURSE.NAME
                     LEVEL COST
                     ID.SUP

One record listed.
```

Example 3-7. A Sample @ Phrase

Note

Note that the ID.SUP keyword is included in the phrase to suppress INFORM's default record ID value display. ID.SUP is documented in Section 4.

When this form of LIST is given:

```
LIST COSTS
```

the display looks like that shown in Example 3-8.

```
:LIST COSTS

LIST COSTS 17:15:02  08-16-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD     COURSE COST

CS101         INTRO TO C.S.        U                 $275.00
MG101         INTRO TO MGMT        U                 $275.00
CS579         DATA BASE CONCEPTS   G                 $475.00
CS673         SOFTWARE ENG         G                 $475.00
CS600         SYSTEMS OPERATION    G                 $475.00
CS301         DATA STRUCTURES      U                 $300.00

6 records listed.
```

Example 3-8. LIST Using an @ Phrase

If no @ phrase exists, the display would look like the one shown in Example 3-9.

```
:LIST COSTS

LIST COSTS 15:26:06  08-17-81  PAGE     1
COSTS.....

CS101
MG101
CS579
CS673
CS600
CS301

6 records listed.
```

Example 3-9. Display Without an @ Phrase

If an @LPTR Phrase Exists: If the LIST sentence contains the LPTR keyword, which directs the display to the line printer instead of the terminal, and no display names are indicated, INFORM looks for the @LPTR phrase and uses the names specified there. If no @LPTR phrase is found, INFORM uses the @ phrase, assuming one exists.

If a Field Name is Specified: If at least one field name, or a phrase which specifies field names, is included in a LIST sentence, the data in the named fields will be displayed. The record ID field is also displayed by default. For example, to produce a list of all the first names and last names of students in the STUDENTS file, use the sentence shown in Example 3-10.

```
:LIST STUDENTS FNAME LNAME

LIST STUDENTS FNAME LNAME 15:27:11  08-17-81  PAGE     1
SOC-SEC-NO.    FIRST NAME.....    LAST NAME.....

141-05-9988    SUSAN              KATZ
301-45-9817    MARVIN             MAYER
412-05-7716    ELAINE             GOUDREAUX
343-05-9988    MARTHA             GRANIPP
131-43-5670    ADELLE             JARVIS
001-45-6677    ROBERT             MACLEAN
111-45-5566    STEVEN             MCLEAN

7 records listed.
```

Example 3-10. Output in Default Column Format

INFORM does not use the @ and @LPTR phrases if at least one field name, or a phrase which specifies field names, appears in the sentence. The exception to this is when @ or @LPTR themselves are named in a sentence.

Default Output: Terminal output is formatted into one or more aligned columns if the combined width of the data to be displayed permits. Most terminal screens allow for a maximum of 80 characters to be displayed per line, although INFORM always allows for a certain number of spaces between columns for readability.

Vertical Display Format: In the event that the data is too wide to be displayed across a single terminal or line printer page, the data will be output in rows, with the display names left-justified on the screen and the data for that field appearing on the same line. This is called INFORM's vertical display mode. (To force any display to be done in vertical mode, use the VERTICALLY keyword, described in Section 4.)

Example 3-11 shows what happens when there is too much data to be displayed in INFORM's usual column format. Since there is too much data to put neatly in readable columns across the page, INFORM lists out the fields using its vertical display format.

```
:LIST STUDENTS FNAME MI LNAME GRAD.YR COURSES GRADES GPA WITH LNAME EQ KATZ

LIST STUDENTS FNAME MI LNAME GRAD.YR COURSES GRADES GPA WITH LNAME EQ KATZ
 20:48:18  08-20-81  PAGE     1
SOC-SEC-NO. 141-05-9988
FIRST NAME. SUSAN
MID. INIT.. J.
LAST NAME.. KATZ
GRAD YEAR..
COURSE NUMBER COURSE GRADE
CS101         A
GPA.....  4.00

One record listed.
```

Example 3-11. Output in Vertical Format

Note

Members of an association are output on the same line in a vertical format, as shown in the previous example. The COURSES and GRADES fields are members of an association called COURSES.GRADES. That way, all associated field values are displayed side-by-side so that their relationship is evident.

More on Output Format: Output formatting is an involved subject that requires further discussion. See OUTPUT KEYWORDS in Section 4 for a complete description of all the output keywords and examples of how to use them.

Selection Expressions

A selection expression is the part of an INFORM sentence that indicates one or more conditions which must be satisfied in order for a record to be selected. Sometimes selection expressions are referred to as search expressions, file interrogations, or file queries. The complexity of a search expressions can range from a simple match condition like:

```
LNAME EQ LADD
```

to a more complex logical expression, such as:

```
LNAME GT LADD AND SALARY LT 24000 OR LEVEL EQ E5
```

Sentences With Selection Criteria: Selection criteria in an INFORM sentence must be preceded by an INFORM verb like LIST, SORT, COUNT, or SUM. The verb must be immediately followed by a filename. The next item in the sentence can be either a list of field names (also called display names), or the selection criteria clause itself. (Recall the general INFORM sentence format in Section 1, Figure 1-1.)

The selection criteria clause is always introduced by one of the selection keywords WITH or WHEN. This keyword is then followed by a selection expression. Sentences which contain selection criteria clauses are generally formatted in this manner:

```
verb filename {WITH | WHEN} [EVERY] selection.expression
```

Any sentence that contains selection criteria must include either the WITH or the WHEN keyword. This is the signal to INFORM that only records meeting certain criteria are to be selected. The WITH keyword is used to limit which records should be chosen on a given search. It can be applied to single-valued and multivalued fields. The WHEN keyword limits the values from certain multivalued fields that will be retrieved on a given search. WHEN can only be used with multivalued fields.

The EVERY keyword chooses only records in which every value of a certain value-marked field meets the stated condition.

The basic format for selection.expression within an INFORM sentence is:

$$\left\{\begin{matrix}\text{WITH}\\\text{WHEN}\end{matrix}\right\}[\text{EVERY}]\ \text{field keyword value}\ \left\{\begin{matrix}\text{AND}\\\text{OR}\end{matrix}\right\}\text{field keyword value ...}$$

The keyword is any one of the relational keywords listed in the table that appears on the following page.

The value is the user-specified value to be used in looking for a match on field in the data file. If no field value is found that fulfills the condition as stated, the search fails and no record values are returned.

Simple Search Conditions: A simple selection expression states a single condition that must be satisfied, such as:

```
LIST STUDENTS WITH LNAME EQ MAYER
```

This expression finds all the records in the STUDENTS file with a last name (LNAME) value of MAYER and displays them.

Complex Search Conditions: Selection expressions can be composed of many conditions, joined by the AND and OR keywords. Only the records that fit all the specified conditions will be selected and displayed.

```
LIST STUDENTS WITH GPA > 3.0 AND GRAD.YR 1985
```

Both parts of this conditional expression must be satisfied in order for any records to be returned.

There are many selection keywords available for use in search expressions. All of INFORM's selection keywords are documented in Section 4 under SELECTION KEYWORDS. Those most commonly used in search expressions are listed in Table 3-8. Synonyms are grouped together and a sample phrase shows how each type of keyword can be used. If you put the words LIST STUDENTS in front of each sample phrase, you have a complete INFORM sentence that can be used with the STUDENTS file that has been used in many of the examples.

Table 3-8
Selection Keywords and Examples

Keyword	Meaning	Example
EQ =	Equal to	WITH GPA EQ 4.0
NE NOT #	Not equal	WITH FNAME NE GRACE
GE >=	Greater than or equal to	WITH GPA GE 2.0
LE <=	Less than or equal to	WITH GRAD.YR LE 1983
AFTER GREATER GT >	Greater than or after in ASCII sequence	WITH LNAME GT KATZ
BEFORE LESS LT <	Less than or before in ASCII sequence	WITH LNAME BEFORE MAYER
MATCHES MATCHING LIKE	Matches pattern	WITH FNAME MATCHING Mc...
UNLIKE NOT.MATCHING	Doesn't match pattern	WITH COURSES NOT.MATCHING ...101
SAID SPOKEN ~	Sounds like	WITH LNAME SAID 'MACLEAN'

Note

"ASCII sequence" refers to the standard ASCII collating sequence. See the INFO/BASIC Reference Guide or Appendix C of the Prime User's Guide for a ranked list of the characters in the ASCII collating sequence.

The MATCHING and NOT.MATCHING keywords, and all their respective synonyms, are used with arguments that may take any one of these three forms:

Format	Meaning
string...	Finds value beginning with string.
...string	Finds value ending with string.
...string...	Finds value with string anywhere in field.

Note

If string contains special characters, like a space or an asterisk (*), the string itself must be enclosed in quotes, as in:

```
WITH PARTNO MATCHING "...'K2-45'..."
```

Both pairs of quotes are needed in this type of search.

Some Sample Selection Expressions

Here are some examples of selection expressions used in LIST sentences. An explanation of what each one does is given first, followed by the actual sentence you would type at INFORM level to produce the results which are shown in the sample displays.

Selections With Selection Keywords: To find and display only those students whose last names begin with the letter K or greater, use the sentence shown in Example 3-12.

```
:LIST STUDENTS FNAME LNAME WITH LNAME AFTER K

LIST STUDENTS FNAME LNAME WITH LNAME AFTER K 15:28:13  08-17-81  PAGE     1
SOC-SEC-NO.    FIRST NAME.....    LAST NAME.....

141-05-9988    SUSAN              KATZ
301-45-9817    MARVIN             MAYER
001-45-6677    ROBERT             MACLEAN
111-45-5566    STEVEN             MCLEAN

4 records listed.
```

Example 3-12. LIST With Selection Clause

Example 3-13 illustrates a LIST sentence that finds course records in the COSTS file in which the course listed costs more than $200.

```
:LIST COSTS WITH COST GT 200

LIST COSTS WITH COST GT 200 15:28:34  08-17-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD     COURSE COST

CS101         INTRO TO C.S.        U                 $275.00
MG101         INTRO TO MGMT        U                 $275.00
CS579         DATA BASE CONCEPTS   G                 $475.00
CS673         SOFTWARE ENG         G                 $475.00
CS600         SYSTEMS OPERATION    G                 $475.00
CS301         DATA STRUCTURES      U                 $300.00

6 records listed.
```

Example 3-13. LIST With Selection Keyword

Selecting a Particular Record: Example 3-14 shows a LIST sentence that locates a particular record in the STUDENTS file given a last name (LNAME) value.

```
:LIST STUDENTS WITH LNAME EQ JARVIS

LIST STUDENTS WITH LNAME EQ JARVIS 15:32:14  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

131-43-5670  ADELLE           JARVIS          CS101          B
                                             CS301          B
                                             CS579          A-
                                             CS673          D
                                             CS673          B+

One record listed.
```

Example 3-14. Listing a Particular Record

Selection With EVERY Keyword: To retrieve records only if every value in a multivalued field matches a condition, use the EVERY keyword. Example 3-15 shows a sentence that will find only those records in the STUDENTS file in which all the entries in the COURSES field begin with CS. This prefix indicates a course in the Computer Science Department.

```
:LIST STUDENTS WITH EVERY COURSES MATCHING CS...

LIST STUDENTS WITH EVERY COURSES MATCHING CS... 15:36:09  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ            CS101          A
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                              CS108          B+
                                              CS216          A
131-43-5670  ADELLE           JARVIS          CS101          B
                                              CS301          B
                                              CS579          A-
                                              CS673          D
                                              CS673          B+

3 records listed.
```

Example 3-15. LIST Using EVERY Keyword

Selection With WHEN Keyword: When you want just certain values from a value-marked field to be displayed, use the WHEN keyword. The next two examples show how the WHEN keyword works. Example 3-16 shows all the values present in the COURSES field for a particular student.

```
:LIST STUDENTS WITH LNAME EQ MAYER

LIST STUDENTS WITH LNAME EQ MAYER 15:36:33  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                             MG101          B+
                                             CS673          A-

One record listed.
```

Example 3-16. LIST Using WHEN Keyword (Part 1)

To display just the CS (Computer Science) courses listed in this record, use the WHEN keyword and the MATCHING keyword, as shown in Example 3-17.

```
:LIST STUDENTS WITH LNAME EQ MAYER WHEN COURSES MATCHING CS...

LIST STUDENTS WITH LNAME EQ MAYER WHEN COURSES MATCHING CS...
15:37:25  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              CS673          A-

2 records listed.
```

Example 3-17. LIST Using WHEN Keyword (Part 2)

Note

The WHEN keyword can only be used with multivalued fields. INFORM displays an error message to that effect if you use it incorrectly.

Sort Expressions

The difference between sort and selection criteria is that selection criteria tells INFORM which records to retrieve and sort criteria tells INFORM by which fields in these records the output should be sorted. Sort criteria is specified in the form of a sort expression in an INFORM sentence. A sort expression consists of at least one of the sort keywords and the field name by which the output records should be sorted. The sort keywords are listed in Table 3-9.

Table 3-9
The Sort Keywords

Keyword	Sorts by:
BY field.name	Ascending order by field.name.
BY.DSND field.name	Descending order by field.name.
BY.EXP field.name	Ascending order within a value-marked field.
BY.EXP.DSND field.name	Descending order within a value-marked field.

Multiple sort keywords and phrases can appear in a single INFORM sentence, as shown in the examples that appear on the following pages.

Some Sample Sort Expressions: To obtain a list of records sorted on the last name field in the STUDENTS file, use the form of LIST shown in Example 3-18.

```
:LIST STUDENTS FNAME LNAME GPA BY LNAME

LIST STUDENTS FNAME LNAME GPA BY LNAME 15:38:28  08-17-81  PAGE     1
SOC-SEC-NO.    FIRST NAME.....    LAST NAME.....    GPA..

412-05-7716    ELAINE             GOUDREAUX         3.43
343-05-9988    MARTHA             GRANIPF           0.00
131-43-5670    ADELLE             JARVIS            2.80
141-05-9988    SUSAN              KATZ              4.00
001-45-6677    ROBERT             MACLEAN           3.85
301-45-9817    MARVIN             MAYER             3.33
111-45-5566    STEVEN             MCLEAN            3.00

7 records listed.
```

Example 3-18. LIST With Sort Clause

Example 3-19 illustrates a LIST sentence that specifies reverse sorting on the GPA field.

```
:LIST STUDENTS LNAME COURSES.GRADES BY.DSND GPA GPA

LIST STUDENTS LNAME COURSES.GRADES BY.DSND GPA GPA 15:34:45  10-21-81  PAGE
  1
SOC-SEC-NO.    LAST NAME.....    COURSE NUMBER    COURSE GRADE    GPA..

141-05-9988    KATZ              CS101            A               4.00
                                 MK101            A
                                 MG101            A
                                 MK107            A
343-05-9988    GRANIFF           SP101            B               3.43
                                 MK101            A
                                 MK108            B+
412-05-7716    GOUDREAUX         CS105            B               3.43
                                 CS108            B+
                                 CS216            A
111-45-5566    MCLEAN            MG101            B               3.33
                                 MK101            A
                                 EN201            B
301-45-9817    MAYER             CS101            B               3.33
                                 MG101            B+
                                 CS673            A-
131-43-5670    JARVIS            CS101            B               3.00
                                 CS301            A
                                 CS579            A-
Press <NEW LINE> to continue...
LIST STUDENTS LNAME COURSES.GRADES BY.DSND GPA GPA 15:34:45  10-21-81  PAGE
  2
SOC-SEC-NO.    LAST NAME.....    COURSE NUMBER    COURSE GRADE    GPA..

                                 CS673            D
                                 CS673            B+

6 records listed.
```

Example 3-19. Reverse Sorting With LIST

Multivalued Sort Keys: The BY.EXP and BY.EXP.DSND keywords are used in sorting multivalued fields. (EXP stands for exploding and DSND for descending.) With these keywords, you can sort the values in a given value-marked field in ascending or descending order. All the values in the named multivalued fields are treated as sort keys. Each multivalued field occurrence is displayed separately, along with the record ID and any other field in the record that must be displayed. For example, in the STUDENTS file, sorting the multivalued field COURSES with the BY.EXP keyword results in the display of Example 3-20.

```
:LIST STUDENTS BY.EXP COURSES

LIST STUDENTS BY.EXP COURSES 15:39:55  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

343-05-9988  MARTHA           GRANIPP
001-45-6677  ROBERT           MACLEAN         CS101          A
131-43-5670  ADELLE           JARVIS          CS101          B
301-45-9817  MARVIN           MAYER           CS101          B
141-05-9988  SUSAN            KATZ            CS101          A
412-05-7716  ELAINE           GOUDREAUX       CS105          B
412-05-7716  ELAINE           GOUDREAUX       CS108          B+
412-05-7716  ELAINE           GOUDREAUX       CS216          A
131-43-5670  ADELLE           JARVIS          CS301          B
131-43-5670  ADELLE           JARVIS          CS579          A-
131-43-5670  ADELLE           JARVIS          CS673          B+
131-43-5670  ADELLE           JARVIS          CS673          D
301-45-9817  MARVIN           MAYER           CS673          A-
001-45-6677  ROBERT           MACLEAN         EN203          A-
111-45-5566  STEVEN           MCLEAN          MG101          B
301-45-9817  MARVIN           MAYER           MG101          B+

16 records listed.
```

Example 3-20. LIST With BY.EXP Keyword

If you want to sort the values in a multivalued field within each record, use a single-valued field as the primary sort key and the multivalued field as a secondary sort key. For example, sort the STUDENTS file on the last name field as a primary sort key, and specify the BY.EXP keyword with the COURSES field as a secondary key, as shown in Example 3-21.

```
:LIST STUDENTS BY LNAME BY.EXP COURSES

LIST STUDENTS BY LNAME BY.EXP COURSES  15:40:44  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

412-05-7716  ELAINE           GOUDREAUX       CS105          B
412-05-7716  ELAINE           GOUDREAUX       CS108          B+
412-05-7716  ELAINE           GOUDREAUX       CS216          A
343-05-9988  MARTHA           GRANIFF
131-43-5670  ADELLE           JARVIS          CS101          B
131-43-5670  ADELLE           JARVIS          CS301          B
131-43-5670  ADELLE           JARVIS          CS579          A-
131-43-5670  ADELLE           JARVIS          CS673          B+
131-43-5670  ADELLE           JARVIS          CS673          D
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
001-45-6677  ROBERT           MACLEAN         EN203          A-
301-45-9817  MARVIN           MAYER           CS101          B
301-45-9817  MARVIN           MAYER           CS673          A-
301-45-9817  MARVIN           MAYER           MG101          B+
111-45-5566  STEVEN           MCLEAN          MG101          B

16 records listed.
```

Example 3-21. Sorting Multivalued Fields

The SORT Verb

SORT and LIST are very much the same except that SORT automatically sorts output records by record ID values. LIST outputs records in sequential order according to how they're stored in the file, unless one of the sort keywords, like BY, BY.DSND, and so forth, is included in the sentence.

The format of the SORT verb is:

SORT [DICT] filename [field names] [selection criteria] [sort criteria] [output options] [record IDs]

These parameters have the same meaning as those described for the LIST verb earlier in this section.

SORT Verb Defaults

Like LIST, if no options are specified with the SORT verb, SORT displays all the record IDs for the indicated file, sorted by record ID value. When no field names are included on the SORT verb line, SORT, like LIST, uses the @ phrase to determine which fields should be displayed. If an @ phrase does not exist in the file dictionary, then only the record ID values will be displayed.

SORT vs. LIST Example: Example 3-22 shows what happens when the COSTS file is listed with the default form of LIST. Example 3-23 shows the same sentence using SORT in place of LIST. Note that there is an @ phrase defined for this file, which specifies these fields: COURSE.NAME, LEVEL, and PRICE.

```
:LIST COSTS

LIST COSTS 15:41:04  08-17-81  PAGE     1
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS101        INTRO TO C.S.           U              $275.00
MG101        INTRO TO MGMT           U              $275.00
CS579        DATA BASE CONCEPTS      G              $475.00
CS673        SOFTWARE ENG            G              $475.00
CS600        SYSTEMS OPERATION       G              $475.00
CS301        DATA STRUCTURES         U              $300.00

6 records listed.
```

Example 3-22. Default LIST Verb Output

```
:SORT COSTS

SORT COSTS 15:41:13  08-17-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD     COURSE COST

CS101         INTRO TO C.S.        U                 $275.00
CS301         DATA STRUCTURES      U                 $300.00
CS579         DATA BASE CONCEPTS   G                 $475.00
CS600         SYSTEMS OPERATION    G                 $475.00
CS673         SOFTWARE ENG         G                 $475.00
MG101         INTRO TO MGMT        U                 $275.00

6 records listed.
```

Example 3-23. Default SORT Verb Output

Note

Unless one of the sort keywords is specified in a SORT sentence, the output will be sorted by ascending record ID value. All sorting is done by ascending value unless otherwise indicated.

SORT Examples

The files used in the following examples are the STUDENTS and COSTS files, both of which have @ phrases defined for default display. The @ phrase in the STUDENTS file defines the following fields: FNAME, LNAME and COURSES.GRADES. The @ phrase of the COSTS file defines the COURSE.NAME, LEVEL, and PRICE fields.

In Example 3-24, the file is sorted by the LEVEL field, which indicates whether the course is at the Graduate level (G), or the Undergradute level (U). Because there is an @ phrase defined for the file, the fields named in the @ phrase are displayed.

```
:SORT COSTS BY LEVEL

SORT COSTS BY LEVEL 15:41:27  08-17-81  PAGE     1
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS579        DATA BASE CONCEPTS      G              $475.00
CS600        SYSTEMS OPERATION       G              $475.00
CS673        SOFTWARE ENG            G              $475.00
CS101        INTRO TO C.S.           U              $275.00
CS301        DATA STRUCTURES         U              $300.00
MG101        INTRO TO MGMT           U              $275.00

6 records listed.
```

Example 3-24. SORT With Sort Clause

Example 3-25 sorts the COSTS file by the PRICE field but only the record with the indicated record ID values will be displayed.

```
:SORT COSTS BY COST "CS101" "CS301"

SORT COSTS BY COST "CS101" "CS301" 15:42:43  08-17-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD     COURSE COST

CS101         INTRO TO C.S.        U                 $275.00
CS301         DATA STRUCTURES      U                 $300.00

2 records listed.
```

Example 3-25. SORT With Record IDs

Example 3-26 sorts the file by LEVEL (ascending) and by descending COURSE.NO.

```
:LIST COSTS BY LEVEL BY.DSND COURSE.NO

LIST COSTS BY LEVEL BY.DSND COURSE.NO 15:35:13  10-21-81  PAGE     1
COURSE NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS673        SOFTWARE ENG            G              $475.00
CS600        SYSTEMS OPERATION       G              $475.00
CS579        DATA BASE CONCEPTS      G              $475.00
MG101        INTRO TO MGMT           U              $275.00
CS301        DATA STRUCTURES         U              $300.00
CS108        PASCAL PROGRAMMING      U              $300.00
CS105        ASSEMBLY LANG           U              $350.00
CS101        INTRO TO C.S.           U              $275.00

8 records listed.
```

Example 3-26. SORT With BY.DSND Keyword

Output Options

The output options parameter indicated in both the LIST and SORT verb formats covers the host of INFORM level keywords available for use with these verbs. The keywords described as Output or Report keywords in Section 4, Table 4-1, can be used with either the LIST or SORT verbs to do a variety of things, such as:

- Forcing suppression of record ID values in displays.
- Adjusting column size and spacing between columns.
- Finding averages or total values for numeric fields.
- Putting headings on reports.
- Double-spacing output records.

See Section 4 for details and examples.

SELECT LIST OPERATIONS

To simplify multiple record processing, INFORM allows you to create lists of record ID values which can then be made available to INFORM level verbs, to certain PERFORM verbs, to the Editor, and to INFO/BASIC programs. These lists are called select lists because they represent a select group of records which met some user-stated search criteria.

About Select Lists

Select lists are created by the SELECT or SSELECT verbs, which specify the criteria to be used in selecting certain records. SSELECT is the same thing as SELECT, except that the record ID values are selected in sorted order. The primary advantage of select lists is that they can be saved and reused in multiple operations, so that a series of operations can be performed on the same set of records without having to recreate the set each time. It means that you access the file once, select the records you want, save the list of record IDs, and then reuse it whenever you need it. This saves a lot of time in file searching because the entire file will not have to be searched each time to retrieve the record IDs you want to process.

Creating a Select List

The SELECT and SSELECT verbs have exactly the same format:

```
{SELECT }[DICT] filename [selection] [ sort    ] [record]
{SSELECT}                [criteria ] [criteria] [ IDs   ]
```

All you need to create a select list are one of the select verbs and a filename. If no selection criteria or record IDs are supplied, the entire contents of the named file will be selected. If the SELECT verb is used without any sort criteria, record IDs will be saved in sequential order according to the way they are stored. If sort criteria are specified, then the saved list will be sorted accordingly. SSELECT without any additional sort criteria will sort the record ID values in ascending order by value, just as the SORT verb does. When the select list is activated, the records will be processed in the order in which their record ID values appear in the list.

Sample Select Lists: Example 3-27 shows how to create a select list of all the records in the COSTS file which contain information about graduate level courses. Such courses are indicated by the letter G in the LEVEL field.

```
:SELECT COSTS WITH LEVEL EQ G

3 records selected.
```

Example 3-27. Forming a Select List

This selected group of records can then be passed to an INFO/BASIC program, to the Editor, or to another INFORM or PERFORM verb, as long as that verb or program is executed immediately after the SELECT or SSELECT verb is issued. After a SELECT or SSELECT verb has been executed, the resulting select list is said to be "active," or "in your work space." This simply means that it's immediately accessible by INFORM, PERFORM, the Editor, and INFO/BASIC programs.

For example, with this select list active, typing the sentence:

LIST COSTS

will yield a display of all the graduate level courses shown in Example 3-28.

```
:LIST COSTS

COSTS.....    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS579         DATA BASE CONCEPTS      G             $475.00
CS673         SOFTWARE ENG            G             $475.00
CS600         SYSTEMS OPERATION       G             $475.00

3 records listed.
```

Example 3-28. Display With Active Select List

Note

As soon as any verb subsequent to a SELECT or SSELECT is executed, the select list will no longer be active.

Select List Verbs

In addition to the SELECT and SSELECT verbs, which are INFORM level verbs, several PERFORM level verbs allow you to manipulate select lists once they've been created. They are listed in Table 3-10.

Table 3-10
Select List Verbs

Verb	Function
SAVE.LIST	Names and stores the current select list for future use.
GET.LIST	Retrieves a stored select list from your account.
DELETE.LIST	Removes a saved select list from your account.
FORM.LIST	Makes a select list from a user-created list of record IDs.

Note

In your VOC file there is a special paragraph called EDIT.LIST. It defines a paragraph that enables you to edit the contents of a saved select list without having to type the &SAVEDLISTS& filename every time you wish to see what's in a particular select list. For example, to look at the contents of a saved list called REPORT, you simply have to type:

```
EDIT.LIST REPORT
```

rather than:

```
EDIT &SAVEDLISTS& REPORT000
```

Saving a Select List

Once you've formed a select list with either SELECT or SSELECT, you can save it for further use with the SAVE.LIST verb.

The SAVE.LIST verb has the following format:

```
SAVE.LIST [list.name]
```

A list.name is optional. If you don't supply one, INFORM will supply a default name of the form:

```
&TEMPnn&
```

where nn is the port number which corresponds to your terminal line. This default temporary select list is overwritten each time you specify a SAVE.LIST without a select list name.

Where Select Lists Are Stored

When a SELECT or SSELECT verb is successfully executed, the resulting list of record IDs is kept temporarily in the user's workspace. This select list must be used immediately by the verb or process for which you created it, as it will be destroyed when the next verb is executed. To save this select list, issue the SAVE.LIST verb before issuing any other verb. If you want to use this select list in any subsequent operation, you must recall the list with the GET.LIST verb. This is because a SAVE.LIST operation, like any other operation, removes any active select list from your workspace.

Whenever you issue the SAVE.LIST verb, the select list is stored in the special &SAVEDLISTS& file in your account. &SAVEDLISTS& is a Type 1 file, which is like a PRIMOS sub-ufd. Each time you store a select list, it is added as another record to this file. Select list names are actually record IDs for the &SAVEDLISTS& file.

Select List Names: User-specified names are automatically stored with a three-digit number appended to them. If the select list is not sorted or if it contains fewer than 3000 entries, the name will have three 0's appended to it, as in:

```
GRADE.REPORT000
```

Whenever a large, sorted select list is saved, it is divided into two or more parts which are numbered 001 through nnn, depending on the total number of records the list contains.

For example, if you issued the verb:

SAVE.LIST INVENTORY.LIST

and the sorted select list contained more than 3000 records, there would be two entries created in the &SAVEDLISTS& file with these names:

INVENTORY.LIST001 INVENTORY.LIST002

The dividing up of records and the appending of numbers to the names is done automatically by the system.

Note

Although INFORMATION appends numbers to the select list names, you don't need to know what they are in order to access them. Simply supply the name that you assigned and the appropriate list will be retrieved.

Retrieving a Select List

Recalling a saved select list requires the GET.LIST verb. This verb loads the appropriate select list into the user's workspace so it can be used by the next verb. The GET.LIST format is:

GET.LIST [[account.name] [list.name]]

If you use the default form of GET.LIST (no arguments), the &TEMPnn& record, if one exists, will be retrieved. nn corresponds to your port number, which is the number assigned to your terminal line.

If you supply an account name, you must supply the name of the select list you want to retrieve also. This allows you to get select lists out of other accounts, assuming you know the names of the select lists you want to retrieve.

If you don't supply an account name, the named select list will be sought in your current account.

What To Do With Select Lists

Once a select list has been formed or retrieved and is active, it can be used in:

INFO/BASIC Programs	Records in the select list can be used in INFO/BASIC programs via the INFO/BASIC READNEXT statement. See the INFO/BASIC Reference Guide for help.
INFORM sentences	If a list of record IDs is not supplied with an INFORM sentence, the IDs from an active select list will automatically be used instead.
PERFORM verbs	If a PERFORM verb doesn't specify any record IDs, the active select list will be used automatically.
The INFORMATION Editor	The Editor automatically looks for a select list to work on whether or not IDs are specified on the ED verb line. If both exist, the select list is processed first, then the records whose IDs appeared in the sentence are processed. If neither are specified, ED automatically prompts for a record ID. ED always prompts for a record ID once the select list or ID list has been processed.

Note

Remember, in order to use a select list in any of these situations, the sentence or program must be executed immediately after the select list is activated.

Select Lists With Exploded Values

If a select list is formed using the BY.EXP or BY.EXP.DSND keywords, the record ID values are stored a bit differently. The general format of what is stored is:

Record ID}multivalue position}subvalue position

A multivalue position indicates the relative location of a particular value in the exploded multivalued field. A subvalue position indicates the relative position of the subvalue within the multivalue position. If no subvalues exist in the field, this position will always be 1.

For example, a select list can be made of all the record IDs which correspond to students averaging a GPA of 3.0 or better. This list can be saved for future use with the SAVE.LIST verb as shown in Example 3-29.

```
:SELECT STUDENTS BY.EXP COURSES WITH GPA GT 3.0

9 records selected.
:SAVE.LIST GPA.LIST
```

Example 3-29. SELECT on a Multivalued Field

The record IDs are stored in GPA.LIST as shown in Example 3-30.

```
:EDIT.LIST GPA.LIST
9 lines long.

----: F2
0001: 001-45-6677}1}1
0002: 301-45-9817}1}1
0003: 141-05-9988}1}1
0004: 412-05-7716}1}1
0005: 412-05-7716}2}1
0006: 412-05-7716}3}1
0007: 301-45-9817}3}1
0008: 001-45-6677}2}1
0009: 301-45-9817}2}1
Bottom at line 9
----:
```

Example 3-30. An Exploded Select List

Notice that a separate entry is created for each of the course number entries in the COURSE field for each record. Since each of the course number entries contains no subvalues, the subvalue position is indicated as 1 for each of the stored select list entries. This list can be recalled for use, as shown in Example 3-31.

```
:GET.LIST GPA.LIST
9 records selected.
```

Example 3-31. Recalling a Select List

Using this active select list in a LIST sentence, for example, results in the output shown in Example 3-32.

```
:LIST STUDENTS

LIST STUDENTS 15:46:14  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

001-45-6677  ROBERT           MACLEAN         CS101          A
301-45-9817  MARVIN           MAYER           CS101          B
141-05-9988  SUSAN            KATZ            CS101          A
412-05-7716  ELAINE           GOUDREAUX       CS105          B
412-05-7716  ELAINE           GOUDREAUX       CS108          B+
412-05-7716  ELAINE           GOUDREAUX       CS216          A
301-45-9817  MARVIN           MAYER           CS673          A-
001-45-6677  ROBERT           MACLEAN         EN203          A-
301-45-9817  MARVIN           MAYER           MG101          B+

9 records listed.
```

Example 3-32. Using an Active Select List

Using Select Lists

The select list verbs are typically used in a sequence similar to the one shown here. The steps are numbered in logical order. This is just a suggested verb sequence and does not represent the only manner in which the select list verbs can be used.

1. Use SELECT or SSELECT to form a selected set of records as shown in Example 3-33.

```
:SELECT STUDENTS WITH GPA LT 2.9

2 records selected.
```

Example 3-33. Forming a Conditional Select List (Part 1)

2. The select list is now in your workspace. It will remain there until the next verb is executed. Use it immediately, while it is still active, as shown in Example 3-34.

```
:LIST STUDENTS

LIST STUDENTS 15:49:26  08-17-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

343-05-9988  MARTHA           GRANIFF
131-43-5670  ADELLE           JARVIS          CS101          B
                                              CS301          B
                                              CS579          A-
                                              CS673          D
                                              CS673          B+

2 records listed.
```

Example 3-34. Display With Conditional Select List (Part 2)

If you don't want to use the select list right away, you can name and save it for future use, as in:

```
:SAVE.LIST GPA.LIST
```

The list is now saved in the &SAVEDLISTS& file under the name GPA.LIST.

3. You can recall this list at any time, as in:

```
:GET.LIST GPA.LIST
2 records selected.
```

The GPA.LIST select list is now active in your workspace.

4. Delete this list when it's no longer needed:

```
DELETE.LIST GPA.LIST
```

All of the select list verbs are also covered in more detail in the PERFORM Reference Guide.

Note

Each time you want to use a select list, you must use SELECT to create it, or if it has been saved with SAVE.LIST, you must use the GET.LIST verb to reactivate the list. An active select list is always removed from the user's workspace by the next verb issued, including any of the select list verbs.

Verbs That Use a Select List

When invoked, some INFORM verbs, PERFORM verbs, and the INFORMATION Editor automatically look for an active select list to work on. When a select list is active, that is, when it has just been created or called up with the GET.LIST verb, these verbs will use this select list in their processing. Such verbs include:

COPY
ED (Editor)
ENTRO
GROUP.STAT
LIST
SORT
SPOOL
T.DUMP

The complete list of verbs that use select lists, plus some pointers on using them, can be found in the PERFORM Reference Guide, under Commands Which Use Select Lists.

ARITHMETIC OPERATIONS

The COUNT and SUM verbs perform simple arithmetic operations on selected file records. COUNT simply counts the number of records in a file that meet a specified set of conditions. SUM adds up the values in numeric fields in a file and prints out the total for each field.

The COUNT Verb

The default form of COUNT simply counts up the total number of records currently in the file. The INFORM COUNT verb should not be confused with the INFO/BASIC COUNT function, which can be used in I-descriptor expressions. (See Section 2.) The format of COUNT is:

COUNT [DICT] filename [selection expression]

COUNT requires only a filename to be specified on the verb line. The number of records in the file will then be counted and the total reported. Field names or output options should not be included. However, you can find out how many records met a certain condition by supplying an appropriate selection expression. The number of records matching the conditions stated in the expression will then be displayed.

COUNT Defaults: COUNT does not use the @ phrase and does not display any field names or values. If any field names are supplied they are simply ignored. When no selection criteria are included, COUNT simply prints out the number of records in the file. Here is an example of what the default form of COUNT does:

```
:COUNT COSTS

6 records counted.
```

To count the number of records in the COSTS file records that met a certain condition, use a sentence like this:

```
:COUNT COSTS WITH PRICE EQ 275


2 records counted.
```

Of the six records in the COSTS file, only two of them meet the stated condition.

The SUM Verb

The SUM verb performs a summation of the values in any numeric field in a file. This summation can be done on a certain subset of records in a file, or across all the records in a file. SUM requires that you name the fields you want summed. The SUM verb format is:

SUM [DICT] filename { field.names / @ / phrase.name } [selection criteria] [record IDs]

In this format, the field.names parameter indicates numeric fields which are to be summed. The optional selection criteria parameter narrows down the records whose values are to be summed. If you specify names of nonnumeric fields on a SUM verb line, all nonnumeric values will be ignored.

Note

INFORM attempts to sum the values in any field whose name appears on a SUM verb line. Blanks or nulls are treated as 0's, however, and INFORM will print the sum for these fields as 0 if any blank values are found in them. If there are no blanks or numeric values in a specified field, no SUM information will be printed.

SUM Examples: The simplest form of SUM must specify at least one numeric field which is to be summed. This can be done by directly including one or more numeric fields on the verb line, or by specifying a phrase that includes a numeric field in Field 2. Examples 3-35 and 3-36 show the correct use of the SUM verb in INFORM sentences.

Example 3-35 sums the values in the COST field of the COSTS file.

```
:SUM COSTS COST

SUM COSTS COST 15:51:26  08-17-81  PAGE     1
COSTS.....     COURSE COST

                ==========
TOTALS           $2,275.00

6 records summed.
```

Example 3-35. A SUM Example

The summation can be limited to just those records that meet one or more stated conditions, as in Example 3-36.

```
:SUM COSTS COST WITH COST EQ 275

SUM COSTS COST WITH COST EQ 275-15:51:48  08-17-81  PAGE      1
COSTS.....     COURSE COST
                ==========
TOTALS              $550.00

2 records summed.
```

Example 3-36. SUM With Selection Criteria

Record ID values can be specified so only the records that have these key values will be summed, as shown in Example 3-37.

```
:SUM COSTS COSTS"CS101" "CS301"

SUM COSTS COST "CS101" "CS301" 15:53:33  08-17-81  PAGE     1
COSTS.....     COURSE COST

                ==========
TOTALS              $575.00

2 records summed.
```

Example 3-37. SUM With Record IDs

Only the records corresponding to these key values will be summed.

MAGNETIC TAPE OPERATIONS

The two INFORM level verbs that deal with tape operations are T.DUMP and T.LOAD. T.DUMP and T.LOAD are generally used to transfer one or two files from one account to another if they happen to be on different machines. These verbs are able to deal with INFORMATION file structure and therefore can be used safely to save and restore INFORMATION files. Both of these verbs can be used only if a tape drive has been assigned to the process trying to use them.

There are other INFORMATION verbs that handle tape operations which are implemented as PERFORM verbs (commands). They are listed, and briefly described under Related PERFORM Tape Verbs below. You will find them also documented in the PERFORM Reference Guide along with the rest of the verbs that are part of the PERFORM system.

Note

The PRIMOS utilities MAGSAV and MAGRST can be used in place of T.DUMP and T.LOAD, as long as the user is aware of the limitations of these verbs. MAGSAV and MAGRST cannot process the logical structure of INFORMATION files. Restoring a file from tape that has a different modulus or file type than the file on the disk could corrupt the logical integrity of the file.

Related PERFORM Tape Verbs

In addition to the T.DUMP and T.LOAD verbs that are documented in this book, there are seven auxiliary magnetic tape verbs available at the PERFORM level. They are listed in Table 3-11.

Table 3-11
PERFORM Magnetic Tape Verbs

Verb	Function
T.ATT	Gives requestor exclusive use of tape drive.
T.BCK [nn]	Backs up tape nn records, or to end-of-file mark.
T.DET	Releases previously assigned tape drive. It is the opposite of T.ATT.
T.FWD [nn]	Advances tape to end-of-file marker or advances tape nn records.
T.READ	Reads and displays next tape record. Use LPTR option to print record instead.
T.REW	Rewinds the tape to the load point.
T.WEOF	Writes an end-of-file mark at current tape position.

All of these verbs recognize the MTU keyword which can be followed by the tape drive number, 7- or 9-track indicator, and an optional mode indicator. These verbs are described in greater detail in the PERFORM Reference Guide.

The T.DUMP Verb

T.DUMP is used to write files from disk to tape. After each T.DUMP operation, INFORM writes an end-of-file marker at the current tape position. Remember, you must first assign a tape drive prior to using the T.DUMP verb. The format is:

T.DUMP [DICT] filename [MTU [mtu]] [BLK nn] [selection criteria] [sort criteria]

The DICT option permits you to save a dictionary file only. The default is to save only the data file. To save both parts of an INFORMATION file, use T.DUMP filename and T.DUMP DICT filename.

If you have more than one tape drive assigned to you at a particular time, or if you have another drive besides unit 0 assigned, you must use the MTU keyword, followed by the appropriate drive number to indicate which drive you want to handle the T.DUMP operation. If you have only one drive assigned, this phrase isn't needed. The mtu parameter is made up of three required parts, as shown in Table 3-12.

Table 3-12
The mtu Parameter

Letter	Indicates	Values	Meaning
m	mode	0	No conversions (ASCII)
		1	EBCDIC conversion
		2	Invert high order bit
t	tracks	0	9-track tape (default)
		1	7-track tape
u	unit number	0-7	Number assigned to tape drive

The default value for mtu is 000, in which the mode is ASCII, tracks is 9-track) and the unit number is 0.

The BLK nn parameter is used to indicate block size, if the desired block size is different from the default, which is 8192 bytes per block. (This is also the maximum block size.)

Note

A tape record and tape block refer to the same thing. They are not the same as a disk record.

The selection criteria and sort criteria phrases should be used when you want only certain records saved on tape. The records will be stored in sorted order if you specify any sort keywords.

End-of-File Marks: T.DUMP automatically writes an end-of-file marker after each individual T.DUMP operation you specify. This makes it easier to read the tape and to advance or rewind it using the T.FWD and T.BCK verbs respectively.

Saving INFORMATION Files on Tape

The first step in dumping files from disk to tape is to mount the tape on an available tape drive. Make sure all the drives that are hooked up to your system are powered up; they don't have to be ON LINE, but the power has to be turned on. The tape should be positioned to the load point by pressing the LOAD button on the tape drive. Then hit the ON LINE button. You can use either the INFORMATION T.ATT verb or the ASSIGN verb to reserve the tape drive for exclusive use. To do this you must know the number assigned to the tape drive. Since most users have only one tape drive, the number is usually 0. This is indicated by the button in the upper left corner of the tape drive.

Assigning the Drive: If you're at PERFORM command level, issue the T.ATT verb in this form:

```
T.ATT [[MTU] mtu]
```

Neither the MTU keyword nor the mtu parameter is required. If the default form is used (no arguments), mtu is assumed to have a value of 000 and device MT0 is assigned. The m represents the mode (ASCII is assumed), the t represents the number of tracks on the tape (7 or 9), and the u represents the unit number of this tape drive. If someone else already "owns" the particular drive you're trying to assign, a message will be displayed indicating that the drive is already in use. Tape drives remain assigned to whomever assigned them until a specific detach request is issued, or until that user logs off the system.

Then you are ready to issue the T.DUMP verb and save the files or records you want. Remember to unassign the drive when you're finished, using the T.DET or UNASSIGN verbs.

Note

Only tape drives which have been assigned with the T.ATT or ASSIGN verb will be recognized in tape operations.

Restoring INFORMATION Files From Tape

Again, a tape drive must be assigned using the T.ATT or the ASSIGN verb before the T.LOAD verb will work. T.LOAD only works on tapes that were created using the T.DUMP verb. Don't attempt to restore tapes created with MAGSAV and MAGRST by using T.LOAD. T.LOAD has the following format:

```
T.LOAD [DICT] filename [MTU] [mtu]] [selection] [OVERWRITING]
                                          criteria
```

The parameters are the same as for the T.DUMP verb. The OVERWRITING keyword is used to overwrite existing disk file records. See Suggestions for Using T.LOAD below.

Note

T.LOAD does not accept the BLK keyword. If BLK is present in a T.LOAD verb, an error message will be displayed indicating that BLK should be used with T.DUMP only.

Suggestions for Using T.LOAD

There are several limitations to be aware of when using T.LOAD:

- T.LOAD accepts a select list of up to 500 record ID values. (The select list must be made active immediately prior to the T.LOAD verb.) If you have a larger list of IDs, it is more efficient to load the entire file into a temporary file (on disk) rather than to try and access it from tape.

- Any sorting criteria indicated in a T.LOAD sentence will be ignored. Use T.DUMP to sort records before they're written to tape.

- The OVERWRITING keyword can be used with T.LOAD to overwrite preexisting records in the disk file. If the keyword is not included in the sentence, preexisting records will not be overwritten. When used with the OVERWRITING keyword, T.LOAD does not read the disk file to see if a record may be overwritten. Instead, it simply writes out every record encountered on the tape without reading the disk file first. This results in improved performance, even if the file is being loaded for the first time (which means it does not contain any records which could be overwritten).

Operations From Magnetic Tape

The INFORM LIST, SELECT, COUNT, and SUM verbs can all use a magnetic tape created with T.DUMP as a source of records. This means that you can perform all the possible operations afforded by these verbs using a magnetic tape instead of a disk file. This is done by including the MTU mtu phrase in the INFORM sentence. However, a dictionary file must be present on disk for every file referenced by these verbs when accompanied by the MTU keyword. For example, if the sentence was:

```
LIST STUDENTS WITH LNAME GT K MTU 001
```

INFORM would look on disk for the dictionary, D_STUDENTS, and would search the tape at its current position for records meeting this condition. T.DUMP does not know how to search for files by name, but instead reads the tape from the current position to the next end-of-file mark, and considers this to be one logical file. Thus, the tape must be positioned to the proper spot in order for such a search operation to be successful.

Restrictions: With the following restrictions, all keywords normally associated with the INFORM verbs listed above can be used in magnetic tape operations.

- All sorting operations, like BY and BY.EXP, are ignored. (Sorting can be done only during a T.DUMP.)
- Active select lists may have a maximum of 500 record IDs.
- You should know the format of your tape when issuing verbs that access a tape instead of a disk file. INFORM looks on disk for the dictionary of the named file and then reads the next file on the tape instead of the corresponding data file on disk. You must position the tape to the correct file in order to achieve the desired results.

SECTION 4

INFORM KEYWORDS

INTRODUCTION

INFORM's fifty plus keywords are the powerhouse of the INFORMATION query language and report generator. This section describes the meaning, use, and syntax of these keywords. Keywords have been divided into the following groups, by function, and are discussed in this order:

- ENTRO Keywords — for data entry validation and verification
- Selection Keywords — for establishing search criteria
- Select List Keywords — for use with select lists
- Sort Keywords — for sorting records on one or more fields
- Output Keywords — for changing INFORM display defaults
- Report Keywords — for including options in reports
- Miscellaneous Keywords — for enhancing various INFORM operations

For your convenience, all the INFORM keywords are listed alphabetically in Table 4-1. A brief description is given of each one. Table 4-2 lists all of the INFORM keywords according to function or type and groups together all of the synonyms for each particular keyword.

Note

A complete list of keywords currently supported on your system can be obtained any time by typing:

```
LIST VOC WITH TYPE EQ K
```

or

```
LISTK
```

at PERFORM command level. To get a hard copy of this list, simply append the LPTR keyword to the end of the above sentence. Notice that all the keywords defined on the system, plus any synonyms you may have created for yourself, will be listed by such a sentence. This means that you may see some keywords that are not defined in the tables below.

Table 4-1
Alphabetical Table of Keywords

Keyword	Type	Description
#	Selection	Not equal
&	Selection	And (Boolean)
<	Selection	Less than
<=	Selection	Less than or equal
=	Selection	Equal to
=<	Selection	Less than or equal
=>	Selection	Greater than or equal
>	Selection	Greater than
>=	Selection	Greater than or equal
AFTER	Selection	Greater than
AND	Selection	And (Boolean)
AVERAGE	Report	Calculates average for field
AVG	Report	Calculates average for field
BEFORE	Selection	Less than
BLK	Misc	Specifies block size for T.DUMP
BREAK-ON	Report	Specifies breakpoint in report
BREAK.ON	Report	Specifies breakpoint in report
BREAK.SUP	Report	Suppresses BREAK.ON display
BY	Sort	Sorts on ascending values
BY-DSND	Sort	Sorts on descending values
BY.DSND	Sort	Sorts on descending values
BY.EXP	Sort	Sorts on ascending multi-values
BY.EXP.DSND	Sort	Sorts on descending multi-values
CALC	Report	Specifies TOTAL calculations
CALCULATE	Report	Specifies TOTAL calculations
COL-SUPP	Output	Suppresses default column headings
COL.SPACES	Report	Specifies inter-column spacing
COL.SPCS	Report	Specifies inter-column spacing
COL.SUP	Output	Suppresses default column headings
DBL-SPC	Output	Double-spaces output records
DBL.SPC	Output	Double-spaces output records
DET-SUPP	Report	Displays breakpoints only
DET.SUP	Report	Displays breakpoints only
DICT	Misc	Uses dictionary instead of data file
EQ	Selection	Equal to
EQUAL	Selection	Equal to
EVERY	Selection	For multi-valued fields
FIRST	Misc	Processes first nn records of file
FOOTING	Report	Defines footer for report page
GE	Selection	Greater than or equal
GREATER	Selection	Greater than
GT	Selection	Greater than
HDR-SUPP	Output	Suppresses default heading
HDR.SUP	Output	Suppresses default heading
HEADER	Report	Overrides default heading
HEADING	Report	Overrides default heading
ID-SUP	Output	Suppresses record ID display

Table 4-1 Continued

ID.ONLY	Report	Displays only record IDs
ID.SUP	Output	Suppresses record ID display
INQUIRING	Selection	INFORM prompts for record IDs
LE	Selection	Less than or equal
LESS	Selection	Less than
LIKE	Selection, ENVIRO	Looks for pattern match
LPTR	Output	Prints output on line printer
LT	Selection	Less than
MARGIN	Report	Specifies margin size
MATCHES	Selection, ENVIRO	Looks for pattern match
MATCHING	Selection, ENVIRO	Looks for pattern match
MTU	Misc	Specifies tape drive unit
NE	Selection	Not equal
NEXT.AVAILABLE	ENVIRO	Uses next sequential ID value
NO.PAGE	Output	Suppresses page pause
NOPAGE	Output	Suppresses page pause
NOT	Selection	Not equal
NOT.MATCHING	Selection	Not matching
ONLY	Report	Displays only record IDs
OR	Selection	Or (Boolean)
OVERWRITING	Misc	T.LOAD overwrites existing records
PCT	Report	Calculates percents
PERCENT	Report	Calculates percents
PERCENTAGE	Report	Calculates percents
REQUIRE.SELECT	Selection	Select list required
SAID	Selection	Sounds like
SAMPLE	Misc	Processes first nn records of file
SAMPLED	Misc	Processes every nth record in file
SAVING	Select list	Makes non-key select list
SELECT.ONLY	Select list	Select list required
SPOKEN	Selection	Sounds like
TEMPL	ENVIRO	Uses one of pre-defined ENVIRO processes
TOTAL	Report	Calculates & displays field total
UNIQUE	Select list	Saves unique values only
UNLIKE	Selection	Not matching
USING	ENVIRO	Uses one of pre-defined ENVIRO processes
VERIFIELD	ENVIRO	Displays VERIFILE field value
VERIFILE	ENVIRO	For input verification
VERIFY	ENVIRO	For input verification
VERT	Output	Displays output in vertical format
VERTICALLY	Output	Displays output in vertical format
WHEN	Selection	Selects only certain multi-values
WITH	Selection	Introduces selection criteria clause
~	Selection	Sounds like

Table 4-2
Keywords Listed by Type

Keyword	Type	Description
LIKE MATCHES MATCHING	ENTRO	Checks input for pattern match
NEXT.AVAILABLE		Uses next sequential ID value
TEMPL USING		Uses one of pre-defined ENTRO processes
VERIFIELD		Displays VERIFILE field value
VERIFILE VERIFY		For input verification
#	Selection	Not equal
NE NOT		
& AND		And (Boolean)
< BEFORE LESS LT		Less than
<= =< LE		Less than or equal
= EQ EQUAL		Equal to
=> >= GE		Greater than or equal
> AFTER GREATER GT		Greater than
EVERY		For multi-valued fields
INQUIRING		INFORM prompts for record IDs
LIKE MATCHES MATCHING		Looks for pattern match
NOT.MATCHING UNLIKE		Not matching
OR		Or (Boolean)
SAID SPOKEN ~		Sounds like
WHEN		Displays certain multi-values
WITH		Introduces selection criteria clause
REQUIRE.SELECT SELECT.ONLY	Select List	Select list required
SAVING		Makes non-key select list
UNIQUE		Saves unique values only
BY	Sort	Sorts on ascending values
BY-DSND BY.DSND		Sorts on descending values
BY.EXP		Sorts on ascending multi-values
BY.EXP.DSND		Sorts on descending multi-values

Table 4-2 Continued

COL-SUPP COL.SUP	Output	Suppresses default column headings
DBL-SPC DBL.SPC		Double-spaces output records
HDR-SUPP HDR.SUP		Suppresses default heading
ID-SUP ID.SUP		Suppresses record ID display
LPTR		Prints output on line printer
NO.PAGE NOPAGE		Suppresses page pause
VERT VERTICALLY		Displays output in vertical format
AVERAGE AVG	Report	Calculates average for field
BREAK.ON BREAK-ON		Specifies breakpoint in report
BREAK.SUP		Suppresses BREAK.ON display
CALC CALCULATE		Specifies TOTAL calculations
COL.SPACES COL.SPCS		Specifies inter-column spacing
DET-SUP DET.SUP		Displays breakpoints only
FOOTING		Defines footer for report page
HEADER HEADING		Overrides default heading
ID.ONLY ONLY		Displays only record IDs
MARGIN		Specifies margin size
PCT PERCENT PERCENTAGE		Calculates percents
TOTAL		Calculates & displays field total
BLK	Misc	Specifies block size for T.DUMP
DICT		Uses dictionary instead of data file
FIRST SAMPLE		Processes first nn records of file
MTU		Specifies tape drive unit
OVERWRITING		T.LOAD overwrites existing records
SAMPLED		Processes every nth record in file

ENTRO KEYWORDS

The INFORM keywords that work with the ENTRO processor are designed to help you validate input data. Such validation includes making sure that some field exists in another file (such as a customer code or student ID), or making sure that the data matches some specified pattern. The NEXT.AVAILABLE keyword is used to generate new sequential (numeric) record ID values each time ENTRO is used to add a new record to the file. These keywords are explained in the list that follows.

Note

This is a general note regarding the use of the following ENTRO keywords:

MATCHING (LIKE, MATCHES)
VERIFIELD
VERIFILE (VERIFY)

Whenever these keywords are included in an ENTRO sentence, they must be accompanied by a field name from the dictionary file. This causes ENTRO to prompt only for the record ID and the accompanying field name. The default @ENTRO or @ phrase is disregarded. The sentence must therefore include the names of all the fields for which you want ENTRO to prompt. This includes any fields which you intend to modify as well. When adding a new data record, the field names you omit from the ENTRO sentence will not be displayed and prompted for, causing blanks to be entered for those fields in the new data records. When modifying a data record, you won't be allowed to see or modify any field not specified in the ENTRO sentence.

▶ LIKE

Description: ENTRO checks input for pattern match.

Synonyms: MATCHING, MATCHES

When used with ENTRO, the pattern match keywords restrict input for certain fields to values that match a stated pattern. The field to be validated must immediately precede the MATCHING keyword, and the pattern must immediately follow the MATCHING keyword.

For example, to ensure that only values beginning with CS are allowed as input for the COURSES field, use this sentence:

```
ENTRO STUDENTS COURSES MATCHING "CS..."
```

If you try to enter values for the COURSES field that do not match this pattern, ENTRO will inform you with the message:

```
Invalid format.
```

and will redisplay the prompt for that field.

Note

When using ENTRO with the MATCHING keyword for data entry (as opposed to data modification) you must include the names of all the fields for which you want ENTRO to prompt, whether you want them validated or not. ENTRO will only prompt you for input to the fields named in the sentence.

Example 4-1 shows how MATCHING is used. ENTRO will prompt only for the fields COURSE.NO, COURSE.NAME, and COST. The record ID for this file is COURSE.NO.

```
:ENTRO COSTS COURSE.NO COST COURSE.NAME MATCHING CS...
COSTS  ENTRO.1  18:16:54 23 AUG 1981

COURSE.NO=CH101
Invalid format.
COURSE.NO=MK101
Invalid format.
COURSE.NO=CS108
New record
COURSE COST=300
COURSE NAME=PASCAL PROGRAMMING
COSTS-Screen 1- 18:17:40 23 AUG 1981
 1 COURSE.NO  CS108
 2 COST       300
 3 COURSE.NAM PASCAL PROGRAMMING

CHANGE= (CR)
```

Example 4-1. Using MATCHING With ENTRO

You can specify a match on two or more different patterns by joining them with the backslash (\) symbol. This symbol behaves like a logical "OR". If the input matches any of the patterns specified with MATCHING, it will be accepted. For example, if you wanted the GRAD.YR field to contain either the value 1982 or 1983, you would use the sentence shown in Example 4-2.

```
:ENTRO STUDENTS LNAME FNAME GRAD.YR MATCHING "1982\1983"
STUDENTS  ENTRO.1  18:18:34 23 AUG 1981

SOC-SEC-NO=123-88-9900
New record
LAST NAME=COOK
FIRST NAME=JAMES
GRAD YEAR=1985
Invalid format.
GRAD YEAR=1982
STUDENTS-Screen 1- 18:18:56 23 AUG 1981
 1 RECORD ID  123-88-9900
 2 LNAME      COOK
 3 FNAME      JAMES
 4 GRAD.YR    1982

CHANGE= (CR)
```

Example 4-2. Pattern Matching With ENTRO

▶ MATCHING

Description: ENTRO checks input for pattern match.

Synonyms: LIKE, MATCHES

See LIKE above.

▶ NEXT.AVAILABLE

Description: ENTRO uses next sequential ID value.

The NEXT.AVAILABLE keyword causes ENTRO to generate a new record ID value every time you hit a null line (CR) in response to the record ID prompt during data entry. ENTRO uses the value stored in the &NEXT.AVIALABLE& record in the file dictionary to provide a new record ID value.

The &NEXT.AVAILABLE& RECORD

The first time you specify the NEXT.AVAILABLE keyword with ENTRO, the &NEXT.AVAILABLE& record is added to the dictionary of the file and it is initialized to a value of 1. You can change this value to whatever you want once the record has been created. Use the INFORMATION Editor or ENTRO to modify this record. The value used by ENTRO is stored in Field 2, the LOCATION field, of the &NEXT.AVAILABLE& record.

Note

> The &NEXT.AVAILABLE& record is an X-item, which is a special type of descriptor mentioned earlier in Section 2. This is the only X-item that INFORM recognizes and uses.

How It Works

Each time you invoke ENTRO with the NEXT.AVAILABLE keyword, and you enter a null line (CR) in response to the record ID prompt, ENTRO looks at the value in the &NEXT.AVAILABLE& record, checks that it doesn't exist in the data file already, and uses it as the record ID. ENTRO then increments the value by 1 so that a new value is ready to use the next time one is needed. If the value in Field 2 of the &NEXT.AVAILABLE& record has already been used for a record ID (no duplicate record IDs are allowed), ENTRO adds 1 to the value until a unique one is found.

For example, if you typed the sentence:

```
ENTRO ORDERS NEXT.AVAILABLE
```

each time you hit a null line in response to the record ID prompt, ENTRO will use the value in the &NEXT.AVAILABLE& record of the ORDERS file dictionary as the record ID value for the new record. If any value besides a null line is entered in response to the record ID prompt, ENTRO will not go to the &NEXT.AVAILABLE& record, but will use what you entered instead.

Example 4-3 shows what ENTRO displays when the NEXT.AVAILABLE keyword is used. The current value in the &NEXT.AVAILABLE& record is 4, as shown by listing the contents of the record before invoking ENTRO.

```
:LIST DICT ORDERS &NEXT.AVAILABLE&

LIST DICT ORDERS &NEXT.AVAILABLE& 20:32:41  08-23-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

&NEXT.AVAILABL X    4
E&

One record listed.

:ENTRO ORDERS NEXT.AVAILABLE
ORDERS  ENTRO.1  20:32:55 23 AUG 1981


ENTRY-NO= (CR)
Next available ENTRY-NO is 4 and is also a New record
ENTRY NAME=
```

Example 4-3. Using NEXT.AVAILABLE

The user would then enter a value for the ENTRY NAME field and all other fields specified in the @ or @ENTRO phrases. The example shows what happens when the user supplies a null line (CR) in response to the record ID prompt, which is the ENTRY-NO prompt in this case.

▶ TEMPL

Description: ENTRO uses one of predefined processes.

Synonyms: USING

The TEMPL and USING keywords tell ENTRO to use one of the special processes stored in the ENTRO.PROCESSES file.

The ENTRO.PROCESSES File

The ENTRO.PROCESSES file, located in the ISYS account, contains the following predefined templates for use in building dictionaries, menus, and selectors for menus. The contents of this file are shown in Example 4-4.

```
:LIST ENTRO.PROCESSES

LIST ENTRO.PROCESSES 11:55:57  09-25-81  PAGE     1
PROCESS.....     TITLE.....

$BUILD.DICT              INFORM DICTIONARY DEFINITION
$ENTER.FMENUS            FORMATTED MENU PROCDEFS
$ENTER.MENUS             MENU PROCESS DEFINITIONS
$ENTER.S                 VOC STORED SENTENCE
$ENTER.SELECTOR          VOC SELECTOR ENTRY
$ENTER.SYS.HELP          ENTER/MODIFY SYSTEM HELP FILE
                         RECORDS

6 records listed.
```

Example 4-4. Contents of ENTRO.PROCESSES File

This keyword is used as follows:

ENTRO filename {TEMPL / USING} entro.process

The name of one of the processes shown in Example 4-4 is supplied in the entro.process argument. Note that the prefix dollar signs ($) that appear in the above example should not be supplied when a process name is included in an ENTRO sentence.

▶ USING

Description: ENTRO uses one of predefined processes.

Synonyms: TEMPL

See TEMPL above.

▶ VERIFILE

Description: For input verification with ENTRO.

Synonyms: VERIFY

The VERIFILE and VERIFY keywords are used to validate that input for a given field exists as a record ID on another file. They also can be used to ensure that only ID values that do not exist on another file can be used as record IDs for the entry file. The entry file is the file to which data is being added with ENTRO.

The field name that precedes the VERIFILE keyword is the field in the entry file that is being validated. The argument that immediately follows the VERIFILE keyword is the name of the file whose record ID values will be checked for a match on whatever the user enters as input for the validated field. The general format is:

ENTRO entry.filename verify.field VERIFILE verify.filename

The file to which data is being added or in which data is being modified is entry.filename. The name of the field being validated is verify.field. The name of the file whose record ID field is to be used by ENTRO in validating the input for verify.field is verify.filename.

In the sample sentence below, only courses that are listed in the COSTS file, where COURSE.NO is the record ID field, will be allowed as valid input for the COURSES field.

ENTRO STUDENTS LNAME GRADES COURSES VERIFILE COSTS

Example 4-5 shows what happens when this sentence is executed and a new record is added to the STUDENTS file.

```
:ENTRO STUDENTS LNAME GRADES COURSES VERIFILE COSTS
STUDENTS  ENTRO.1  15:35:06 24 AUG 1981


SOC-SEC-NO=333-45-9900
New record
LAST NAME=GRANT
STUDENTS-Screen 2-COURSES GRADES 15:35:24 24 AUG 1981
RECORD ID==> 333-45-9900
No. COURSES..... GRADES.....
  1
COURSE NUMBER=CS527
CS527 not found on COSTS.
COURSE NUMBER=CS101
COURSE GRADE=B
STUDENTS-Screen 2-COURSES GRADES 15:35:44 24 AUG 1981
RECORD ID==> 333-45-9900
No. COURSES..... GRADES.....
  1 CS101       B
  2
COURSE NUMBER=MG101
COURSE GRADE=A
STUDENTS-Screen 2-COURSES GRADES 15:35:58 24 AUG 1981
RECORD ID==> 333-45-9900
No. COURSES..... GRADES.....
  1 CS101       B
  2 MG101       A
  3
COURSE NUMBER=(CR)
```

Example 4-5. Validating ENTRO Input With VERIFILE

Special Use of VERIFILE

To ensure that only values that do not appear as record ID values on another file are used as record IDs for the entry file, specify the record ID field name of the entry file immediately prior to the VERIFILE keyword. The format for this is:

ENTRO entry.file [field.names] record.id {VERIFILE / VERIFY} verify.file

The record.id field represents the record ID for the entry.file. It must appear just before the VERIFILE or VERIFY keywords. ENTRO looks up any user-entered record ID value for the entry.file in the verify.file. If a match is found, the record ID value will be rejected.

For example, if a temporary file containing new customer orders is being created, it might be necessary to verify that the customer ID field, called CUST.ID, does not match any of the customer ID entries in the master ORDERS file. Both the ORDERS file and the TEMP file use the customer ID field as the record ID.

Using this sentence will prevent the addition of any records to the TEMP file with customer ID fields that are already present in the master ORDERS file:

```
ENTRO TEMP NAME ADDR ORDER.INFO CUST.ID VERIFILE ORDERS
```

Note

When used with the VERIFIELD keyword, this feature of the VERIFILE keyword will not work in the manner just described. (See VERIFIELD below.) Instead, ENTRO will check that the record ID does exist in the VERIFILE file and only then will it be accepted as valid input. In other words, VERIFILE and VERIFIELD used together ensure that only record ID values that already exist in another file will be accepted as record IDs for the entry file.

▶ VERIFIELD

Description: Displays VERIFILE field value.

The VERIFIELD keyword is used in conjunction with the VERIFILE or VERIFY keywords to display one or more fields from the VERIFILE file.

For example, to verify that a course number entry is valid, and to display the name associated with a course number entry, use the sentence:

```
ENTRO STUDENTS COURSES VERIFILE COSTS VERIFIELD COURSE.NAME
```

If the entry for COURSE.NO is not found in the COSTS file, a warning message is displayed. When a valid COURSE.NO entry is input, the COURSE.NAME associated with this entry is displayed just below the COURSE.NO= prompt, as shown in Example 4-6.

```
:ENTRO STUDENTS COURSES VERIFILE COSTS VERIFIELD COURSE.NAME
STUDENTS  ENTRO.1  20:21:07 23 AUG 1981


SOC-SEC-NO=131-48-5765
STUDENTS-Screen 1-FIRST SCREEN 20:21:13 23 AUG 1981
 1 RECORD ID  131-48-5760

S1 == FIRST SCREEN
S2 == COURSES GRADES

CHANGE=S2
STUDENTS-Screen 2-COURSES GRADES 20:21:16 23 AUG 1981
RECORD ID==> 131-48-5760
No. COURSES..... GRADES.....
  1 CS105
  2

Change which line item=2
COURSE NUMBER=EG101
EG101 not found on COSTS.
COURSE NUMBER=CS101
INTRO TO C.S.
STUDENTS-Screen 2-COURSES GRADES 20:21:28 23 AUG 1981
RECORD ID==> 131-48-5760
No. COURSES..... GRADES.....
  1 CS105
  2 CS101
  3
COURSE NUMBER= (CR)
```

Example 4-6. Verifying ENTRO Input Using VERIFIELD/VERIFILE

▶ VERIFY

Description: For input verification with ENTRO.

Synonyms: VERIFILE

See VERIFILE above.

SELECTION KEYWORDS

The selection keywords can be used with the LIST, SORT, SELECT, SSELECT, COUNT, SUM, T.DUMP, and T.LOAD verbs. Some of these keywords have already been introduced in Section 3, but each one is again defined here. The more complex keywords are discussed in greater detail, and examples are given where helpful. Keywords are presented in alphabetical order.

▶ #

Description: Not equal to.

Synonyms: NE, NOT

The "not equal" keywords are used in selection clauses to indicate that field values that are not equal to the indicated criterion should be chosen. The sentence shown in Example 4-7 lists all the records in the students file where the LNAME (last name) field does not equal KATZ.

```
:LIST STUDENTS WITH LNAME NE KATZ

LIST STUDENTS WITH LNAME NE KATZ 10:25:50  09-25-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              MG101          B+
                                              CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                              CS108          B+
                                              CS216          A
131-43-5670  ADELLE           JARVIS          CS101          B
                                              CS301          A
                                              CS579          A-
                                              CS673          D
                                              CS673          B+
001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-
111-45-5566  STEVEN           MCLEAN          MG101          B
343-05-9988  MARTHA           GRANIPF         CS101
                                              CS105
333-45-9900  MIKE             GRANT           CS101          B
                                              MG101          A

7 records listed.
```

Example 4-7. Using the "Not Equal" Keyword

▶ &

Description: Boolean "AND" operator.

Synonyms: AND

When two or more phrases are combined by Boolean ANDs, the field values chosen must meet all the stated conditions. The & and AND keywords are used in selection expressions to combine condition clauses. For instance, the sentence shown in Example 4-8 contains two condition clauses: "NAME GT K" and "COURSES EQ CS101", which are joined by the AND keyword.

```
:LIST STUDENTS FILE WITH LNAME GT K AND COURSES EQ CS101

LIST STUDENTS FILE WITH LNAME GT K AND COURSES EQ CS101 15:15:56  08-21-81  PAGE 1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ           CS101          A
301-45-9817  MARVIN           MAYER          CS101          B
                                             MG101          B+
                                             CS673          A-
001-45-6677  ROBERT           MACLEAN        CS101          A
                                             EN203          A-

3 records listed.
```

Example 4-8. Using the AND Keyword

Only records with LNAME values satisfying both conditions will be chosen.

▶ <

Description: Less than.

Synonyms: BEFORE, LESS, LT

The "less than" keywords can be used with both numeric and nonnumeric fields. Field values are compared on the basis of their rank in the ASCII collating sequence.

In Example 4-9, the sentence shown lists all of the last name values that logically precede MACLEAN in the alphabet.

```
:LIST STUDENTS WITH LNAME BEFORE MACLEAN

LIST STUDENTS WITH LNAME BEFORE MACLEAN 15:45:17  08-21-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ            CS101          A
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                              CS108          B+
                                              CS216          A

343-05-9988  MARTHA           GRANIFF
131-43-5670  ADELLE           JARVIS          CS101          B
                                              CS301          A
                                              CS579          A-
                                              CS673          D
                                              CS673          B+

4 records listed.
```

Example 4-9. The BEFORE Keyword

▶ <=

Description: Less than or equal to.

Synonyms: =<, LE

The "less than or equal to" keywords are used primarily with numeric fields, but can be applied to nonnumeric ones as well. Field values are compared on the basis of their rank in the ASCII collating sequence. Example 4-10 shows the use of the LE keyword.

```
:LIST COSTS WITH COST LE 375

LIST COSTS WITH COST LE 375 15:46:14  08-21-81  PAGE     1
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS101        INTRO TO C.S.           U              $275.00
MG101        INTRO TO MGMT           U              $275.00
CS301        DATA STRUCTURES         U              $300.00

3 records listed.
```

Example 4-10. The LE Keyword

▶ =

Description: Equal to.

Synonyms: EQ, EQUAL

The "equal to" keywords are used with both numeric and nonnumeric fields to indicate that the chosen values must be identical to the stated conditional. In Example 4-11, only the record with an LNAME value of KATZ will be chosen.

```
:LIST STUDENTS WITH LNAME EQ KATZ

LIST STUDENTS WITH LNAME EQ KATZ 15:49:40  08-21-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ            CS101          A

One record listed.
```

Example 4-11. The EQ Keyword

▶ =<

Description: Less than or equal to.

Synonyms: <=, LE

See <= above.

▶ =>

Description: Greater than or equal to.

Synonyms: >=, GE

These keywords are used with both numeric and nonnumeric fields in selection expressions. The sentence in Example 4-12 will retrieve all of the records in the STUDENTS file with a GPA greater than or equal to 3.0.

```
:LIST STUDENTS LNAME GPA WITH GPA GE 3.0

LIST STUDENTS LNAME GPA WITH GPA GE 3.0 15:47:18  08-21-81  PAGE     1
SOC-SEC-NO.    LAST NAME.....    GPA..

141-05-9988    KATZ              4.00
301-45-9817    MAYER             3.33
412-05-7716    GOUDREAUX         3.43
131-43-5670    JARVIS            3.00
001-45-6677    MACLEAN           3.85
111-45-5566    MCLEAN            3.00

6 records listed.
```

Example 4-12. The GE Keyword

▶ >

Description: Greater than.

Synonyms: AFTER, GREATER, GT

The "greater than" keywords are used in selection expressions with both numeric and nonnumeric fields. In Example 4-13, all the last name values that are greater than the characters MA in the ASCII collating sequence are displayed.

```
:LIST STUDENTS WITH LNAME AFTER MA

LIST STUDENTS WITH LNAME AFTER MA 15:47:01  08-21-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              MG101          B+
                                              CS673          A-
001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-
111-45-5566  STEVEN           MCLEAN          MG101          B

3 records listed.
```

Example 4-13. The AFTER Keyword

▶ >=

Description: Greater than or equal to.

Synonyms: =>, GE

See => above.

▶ AFTER

Description: Greater than.

Synonyms: >, GREATER, GT

See > above.

▶ AND

Description: Boolean "AND" operator.

Synonyms: &

See & above.

▶ BEFORE

Description: Less than.

Synonyms: <, LESS, LT

See < above.

▶ EQ

Description: Equal to.

Synonyms: =, EQUAL

See = above.

▶ EQUAL

Description: Equal to.

Synonyms: =, EQ

See = above.

▶ EVERY

Description: Tells INFORM to return a record only if every value in a certain multivalued field meets the specified condition. The EVERY keyword must always be used with the WITH keyword in an INFORM sentence.

In Example 4-14, the sentence shown will list only those records in the STUDENTS file where the GRADES field contains all A's.

```
:LIST STUDENTS WITH EVERY GRADES EQ "A"

LIST STUDENTS WITH EVERY GRADES EQ "A" 16:46:07  10-08-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ            CS101          A
                                              MK101          A
                                              MG101          A
                                              MK107          A

One record listed.
```

Example 4-14. Using the EVERY Keyword

▶ GREATER

Description: Greater than.

Synonyms: <, AFTER, GT

See > above.

▶ GT

Description: Greater than.

Synonyms: <, AFTER, GREATER

See > above.

▶ INQUIRING

Description: Tells LIST to prompt for record IDs.

The INQUIRING keyword is used with the LIST verb. It forces LIST to prompt for the record ID values of the records to be listed. If an existing record ID value is entered, the record will be displayed, using the default @ phrase or the field names specified in the INFORM sentence. If a nonexisting record ID value is entered, a message to this effect is displayed.

Example 4-15 shows how the INQUIRING keyword is used.

```
:LIST COSTS INQUIRING
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

RECORD=CS101
LIST COSTS INQUIRING 20:44:13  08-12-81  PAGE     1
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

CS101        INTRO TO C.S.           U              $275.00
RECORD=MG502
LIST COSTS INQUIRING 20:44:31  08-12-81  PAGE     1
COURSE-NO    COURSE NAME.....    GRAD/UGRAD    COURSE COST

MG502 is not present on COSTS
```

Example 4-15. Using the INQUIRING Keyword

LIST keeps prompting for record IDs until you enter a null line (carriage return) in response to the RECORD= prompt. The @ phrase is automatically used for display names unless you include field names in the LIST sentence.

▶ LESS

Description: Less than.

Synonyms: <, BEFORE, LT

See < above.

▶ LIKE

Description: Looks for pattern match.

Synonyms: MATCHES, MATCHING

LIKE and its synonyms find values that match a certain indicated pattern. The pattern can appear in the beginning, at the end, or anywhere in the string.

The pattern match argument must be specified in one of the following forms:

Format	Meaning
string...	Finds records with fields beginning with string.
...string...	Finds records with fields that contain string anywhere in the field.
...string	Finds records with fields that end with string.

Note

If string contains special characters like an asterisk (*), a hyphen (-), or a blank, then the string portion of the expression must be enclosed in quotes, plus the entire pattern match expression must be enclosed in yet another set of quotes. For example:

```
LIST INVENTORY WITH PARTNO MATCHING "...'MA-50'..."
```

Both sets of quotes are required in this case.

These keywords can also be used with the ENTRO processor to limit the format of entries for a particular field. See ENTRO KEYWORDS earlier in this section for more details.

▶ LT

Description: Less than.

Synonyms: <, BEFORE, LESS

See < above.

▶ MATCHES

Description: Looks for pattern match.

Synonyms: LIKE, MATCHING

See LIKE above.

▶ MATCHING

Description: Looks for pattern match.

Synonyms: LIKE, MATCHES

See LIKE above.

▶ NE

Description: Not equal to.

Synonyms: #, NOT

See # above.

▶ NOT

Description: Not equal to.

Synonyms: #, NE

See # above.

▶ NOT.MATCHING

Description: Looks for fields that do not match pattern.

Synonyms: UNLIKE

The NOT.MATCHING and UNLIKE keywords find records with fields that do not match an indicated pattern. This is the logical opposite of the LIKE, MATCHING, and MATCHES keywords. The same pattern argument format applies for NOT.MATCHING and UNLIKE as for LIKE, MATCHES, and MATCHING, above.

The pattern match argument must be specified in one of the following forms:

Format	Meaning
string...	Finds records with fields that do not begin with string.
...string...	Finds records with fields that do not contain string anywhere in the field.
...string	Finds records with fields that do not end with string.

For example, to find all the students who are taking courses other than computer science (CS) courses, use the sentence in Example 4-16.

```
:LIST STUDENTS WITH EVERY COURSES NOT.MATCHING CS...

LIST STUDENTS WITH EVERY COURSES NOT.MATCHING CS... 16:44:49  10-08-81  PAGE
  1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

111-45-5566  STEVEN           MCLEAN          MG101          B
                                              MK101          A
                                              EN201          B
343-05-9988  MARTHA           GRANIFF         SP101          B
                                              MK101          A
                                              MK108          B+

2 records listed.
```

Example 4-16. The NOT.MATCHING Keyword

Note

UNLIKE and NOT.MATCHING cannot be used with the ENTRO processor.

▶ OR

Description: Boolean "OR" operator.

The OR keyword performs a Boolean OR operation between two conditional clauses. At least one of the conditions has to be satisfied by a single record in order for it to be selected. The OR keyword says that either one, or both, of the stated conditions must be met in order for a selection to occur.

The sentence shown in Example 4-17 finds records from the STUDENTS file with last names beginning with the letter K or greater, or grade point averages of 3.3 or better.

```
:LIST STUDENTS WITH LNAME GT K OR GPA GT 3.3

LIST STUDENTS WITH LNAME GT K OR GPA GT 3.3 10:56:54  09-01-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                             MG101          B+
                                             CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                             CS108          B+
                                             CS216          A
111-45-5566  STEVEN           MCLEAN          MG101          B
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
                                             EN203          A-
333-45-9900  MIKE             GRANT           CS101          B
                                             MG101          A

6 records listed.
```

Example 4-17. The OR Keyword

▶ SAID

Description: Sounds like.

Synonyms: SPOKEN, ~

The "sounds like" keywords find values in the file that match an approximate phonetic spelling of some field value. Spellings should be reasonable approximations of English pronunciations. Example 4-18 shows a common use of these keywords.

```
:LIST STUDENTS WITH LNAME SPOKEN MACLAN

LIST STUDENTS WITH LNAME SPOKEN MACLAN 19:39:48  08-12-81  PAGE     1
SOC-SEC-NO. FIRST NAME..... LAST NAME..... COURSE NUMBER GRADE FOR COURSE

001-45-6677 ROBERT          MACLEAN         CS101         A
                                            EN203         A-
111-45-5566 STEVEN          MCLEAN          MG101         B

2 records listed.
```

Example 4-18. Using the SPOKEN Keyword

The SPOKEN keyword works exactly the same way as the SAID keyword. Example 4-19 shows how.

```
:LIST STUDENTS WITH LNAME SAID MCLAIN

SOC-SEC-NO. FIRST NAME..... LAST NAME..... COURSE NUMBER GRADE FOR COURSE
001-45-6677 ROBERT          MACLEAN         CS101         A
                                            EN203         A-
111-45-5566 STEVEN          MCLEAN          MG101         B

2 records listed.
```

Example 4-19. Using the SAID Keyword

Other common names that can be spelled a number of ways and which could be retrieved using this feature are:

Frederick, Freidrich, Freidrick, and so forth

MacMullen, McMullen

Note

The SAID and SPOKEN keywords will retrieve records only if the first letter of the "sounds like" value matches the first letter of the field value being sought in the data file.

▶ SPOKEN

Description: Sounds like.

Synonyms: SAID, ~

See SAID above.

▶ UNLIKE

Description: Looks for fields that do not match pattern.

Synonym: NOT.MATCHING

See NOT.MATCHING above.

▶ WHEN

Description: Selects only certain multivalued fields.

The WHEN keyword selects only those values in a multivalued field that meet a stated condition or match a certain value. It is a value-limiting keyword, in that it limits the display of values in a multivalued field to those that match a stated condition. This contrasts with the WITH keyword which selects certain records based on stated conditions but cannot limit the display of field values within those records. A record must first meet the selection criteria specified in the WITH clause, before the criteria specified in the WHEN clause can be fulfilled.

Note

Every selection clause in an INFORM sentence must be introduced with either the WITH or WHEN keywords. However, a sentence can contain both keywords.

In the following example, each record in the STUDENTS file lists all the courses a student has taken and the grades for those courses. To see just the courses in which students got A's, use the sentence shown in Example 4-20.

```
:LIST STUDENTS WITH LNAME EQ MACLEAN

LIST STUDENTS WITH LNAME EQ MACLEAN  18:16:45  10-09-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

001-45-6677  ROBERT           MACLEAN         CS101          A
                                             EN203          A-
                                             MK101          B+
                                             CS105          C

One record listed.

:LIST STUDENTS WITH LNAME EQ MACLEAN WHEN GRADES EQ "A"

LIST STUDENTS WITH LNAME EQ MACLEAN WHEN GRADES EQ "A" 18:16:56  10-09-81  PAG
E     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

001-45-6677  ROBERT           MACLEAN         CS101          A

One record listed.
```

Example 4-20. Using the WHEN Keyword

▶ WITH

Description: Introduces selection criteria.

The WITH keyword is a record-limiting keyword, in contrast with the WHEN keyword, which is a value-limiting keyword. WITH selects and displays records on the basis of one or more stated conditions. It cannot selectively display multivalued field occurrences within those chosen records. Note that either the WITH or WHEN keywords must precede any selection expression in an INFORM sentence.

Example 4-21 shows the use of the WITH keyword:

```
:LIST STUDENTS LNAME GPA GRAD.YR WITH GRAD.YR EQ 1984

LIST STUDENTS LNAME GPA GRAD.YR WITH GRAD.YR EQ 1984 18:19:14  10-09-81  PAGE
  1
SOC-SEC-NO.     LAST NAME.....     GPA..     GRAD YEAR

412-05-7716     GOUDREAUX           3.43     1984
001-45-6677     MACLEAN             3.25     1984

2 records listed.
```

Example 4-21. Using the WITH Keyword

▶ ~

Description: Sounds like.

Synonyms: SAID, SPOKEN

See SAID above.

SELECT LIST KEYWORDS

The SAVING and UNIQUE keywords are used with the SELECT verb to create select lists made up of values from nonkey fields. Select lists are usually made up of record ID (key) values. The REQUIRE.SELECT and SELECT.ONLY keywords require an active select list in order for an operation to be performed.

▶ REQUIRE.SELECT

Description: Operation requires a select list.

Synonym: SELECT.ONLY

This keyword can be used with any INFORM verb to specify that the operation cannot take place unless there is an active select list present. Simply append the keyword to the INFORM sentence. In the absence of a select list, a message like the one shown in this example will be displayed:

```
LIST STUDENTS SELECT.ONLY

No active select list was found. Processing terminated
```

Example 4-22 shows what results when a select list has been created or activated by a GET.LIST operation.

```
:SELECT STUDENTS WITH GT.B EQ 3

One record selected.

LIST STUDENTS SELECT.ONLY

LIST STUDENTS SELECT.ONLY 13:41:59  09-06-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

131-43-5670  ADELLE           JARVIS          CS101          B
                                             CS301          A
                                             CS579          A-
                                             CS673          D
                                             CS673          B+

One record listed.

LIST STUDENTS SELECT.ONLY

No active select list was found. Processing terminated.
```

Example 4-22. The SELECT.ONLY Keyword

▶ SAVING

Description: Makes select list of nonkey values.

The SAVING keyword can be used with SELECT to make a select list of field values other than record ID (key) values. Such select lists are useful when saving a field from one file which happens to be the record ID field in another. For example, in a sample file of monthly bills and expenses, one field is the name of the account to which money is owed. Another file stores address information on each of these account names. The first file is called EXPENSES, the second ACCOUNTS. The ACCOUNT-NAME field of the first file is the record ID field for the second file.

The ACCOUNTS file dictionary is listed in Example 4-23.

```
:LIST DICT ACCOUNTS

LIST DICT ACCOUNTS 16:20:09  08-30-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

ACCOUNT-NAME   D    0                    ACCOUNT NAME    30T    S
STREET         D    1                    STREET          20L    S
CITY           D    2                                    15L    S
STATE          D    3                                    4L     S
ZIP            D    4                                    5L
@              PH   ACCOUNT-NAME
                    ADDRESS ID.SUP
ADDRESS        PH   STREET CITY
                    STATE ZIP
@ENTRO         PH   ACCOUNT-NAME
                    ADDRESS

8 records listed.
```

Example 4-23. The ACCOUNTS Dictionary File

The EXPENSES file dictionary is shown in Example 4-24.

```
:LIST DICT EXPENSES

LIST DICT EXPENSES 16:20:17  08-30-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

SERVICE         D   0                    TYPE OF SERVICE 10L    S
ACCOUNT-NAME    D   1                                    30 T   S
ACCOUNT-NO      D   2                                    12L    S
COST            D   3              MD0                   8R2$   M  MONTHLY-CO
                                                                    ST
MONTH           D   4                                    11L    M  MONTHLY-CO
                                                                    ST
MTHLY-TOTAL     I   TRANS(MONTHS,RE MD0                   8R$
                    USE(MONTH),MONT
                    H-NO,"X");IF @1
                    EQ @MONTH THEN
                    TOTAL(COST)
                    ELSE "*****"
@               PH  ACCOUNT-NAME
                    MONTHLY-COST
MONTHLY-COST    PH  MONTH COST
@ENTRO          PH  SERVICE
                    ACCOUNT-NAME
                    ACCOUNT-NO COST
                    MONTH
```

Example 4-24. The EXPENSES Dictionary File

If a SELECT is done on the EXPENSES file, and the ACCOUNT-NAME field is saved (using the SAVING keyword), the address information for these accounts can be retrieved from the ACCOUNTS file as shown in Example 4-25.

```
:SELECT EXPENSES WITH MONTH EQ AUGUST SAVING ACCOUNT-NAME

3 records selected.

:LIST ACCOUNTS

LIST ACCOUNTS 16:20:04  08-30-81  PAGE     1
ACCOUNT NAME..... STREET..... CITY..... STATE ZIP..

NEW ENGLAND TELEPHONE       P.O. BOX 1187      BOSTON      MA    02114
SAM MOLLICA                 234 CLARENDON ST   BOSTON      MA    02116
AMERICAN EXPRESS            P.O. BOX 13767     PHOENIX     AZ    85002

3 records listed.
```

Example 4-25. Using the SAVING Keyword

Note

The SAVING keyword cannot be used with multivalued fields.

▶ SELECT.ONLY

Description: Operation requires a select list.

Synonym: REQUIRE.SELECT

See REQUIRE.SELECT above.

▶ UNIQUE

Description: Saves unique nonkey values only.

This keyword is used with the SAVING keyword to form select lists from unique nonkey values. This prevents duplicate entries in a nonkey select list. Use the UNIQUE keyword as shown in this format:

SELECT filename SAVING UNIQUE field.name

The SAVING keyword is only valid when used with the SAVING keyword. values are saved.

Note

The SAVING keyword causes an implied sort to be done on the SAVING field.name during the selection process, making it illegal to use sort clauses with this keyword. Do not use sort clauses or multivalued field names with the UNIQUE keyword.

SORT KEYWORDS

The SORT keywords can be used with the LIST, SORT, SELECT, SSELECT, and T.DUMP verbs. BY and BY.DSND are intended for use with single-valued fields, while the BY.EXP (EXP stands for exploding) and the BY.EXP.DSND keywords are meant for sorting values in multivalued fields. The general format for using any of these keywords in a sentence is:

```
verb filename { BY          } field.name [ BY          field.name... ]
              { BY.DSND     }            [ BY.DSND                   ]
              { BY.EXP      }            [ BY.EXP                    ]
              { BY.EXP.DSND }            [ BY.EXP.DSND               ]
```

field.name should be single-valued for the BY and BY.DSND keywords, and multivalued for the BY.EXP and BY.EXP.DSND keywords. More than one sort clause can be included in an INFORM sentence. Sort clauses are processed from left to right in a sentence, so the first sort clause will be interpreted as the primary sort key, while all others are secondary.

Fields with an R in the FORMAT field (Field 5) are sorted right-justified, while fields with an L or T in Field 5 are sorted left-justified.

▶ BY

Description: Sorts on ascending values.

The BY keyword can sort both single-valued and multivalued fields. However, multivalued fields are treated as single-valued fields and are sorted as a unit, rather than as individial values, like they are when sorted by the BY.EXP or BY.EXP.DSND keywords. More than one BY field.name clause can be specified in an INFORM sentence. Example 4-26 shows the use of the BY keyword:

```
:LIST COSTS BY COURSE-NO

LIST COSTS BY COURSE-NO 17:14:01  09-24-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD    COURSE COST

CS101         INTRO TO C.S.          U               $275.00
CS108         PASCAL PROGRAMMING     U               $300.00
CS301         DATA STRUCTURES        U               $300.00
CS579         DATA BASE CONCEPTS     G               $475.00
CS600         SYSTEMS OPERATION      G               $475.00
CS673         SOFTWARE ENG           G               $475.00
MG101         INTRO TO MGMT          U               $275.00

7 records listed.
```

Example 4-26. Sorting With BY

▶ BY-DSND

Description: Sorts on descending values.

Synonym: BY.DSND

This keyword works like BY except that field values are sorted in descending order instead of ascending order. Again, multivalued fields will be treated like single-valued fields when sorted with BY.DSND instead of BY.EXP.DSND.

Example 4-27 shows the use of BY.DSND to perform reverse sorting.

```
:LIST COSTS BY.DSND COST

LIST COSTS BY.DSND COST 17:14:22  09-24-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD    COURSE COST

CS600         SYSTEMS OPERATION       G              $475.00
CS673         SOFTWARE ENG            G              $475.00
CS579         DATA BASE CONCEPTS      G              $475.00
CS108         PASCAL PROGRAMMING      U              $300.00
CS301         DATA STRUCTURES         U              $300.00
MG101         INTRO TO MGMT           U              $275.00
CS101         INTRO TO C.S.           U              $275.00

7 records listed.
```

Example 4-27. Reverse Sorting With BY.DSND

▶ BY.DSND

Description: Sorts on descending values.

Synonyms: BY-DSND

See BY-DSND above.

▶ BY.EXP

Description: Sorts on ascending multivalued fields.

This keyword sorts the values within a multivalued field in ascending order. Each multivalue within the field is treated individually, and is displayed with the record ID and any other applicable fields as if it were an entry in a single-valued field. If the name of a single-valued field is specified with this keyword, a warning message is displayed, and the field is simply sorted as a single-valued field.

Example 4-28 uses the BY.EXP keyword:

```
:LIST STUDENTS BY.EXP COURSES

LIST STUDENTS BY.EXP COURSES 17:14:52  09-24-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

333-45-9900  MIKE             GRANT           CS101          B
001-45-6677  ROBERT           MACLEAN         CS101          A
301-45-9817  MARVIN           MAYER           CS101          B
141-05-9988  SUSAN            KATZ            CS101          A
412-05-7716  ELAINE           GOUDREAUX       CS105          B
412-05-7716  ELAINE           GOUDREAUX       CS108          B+
412-05-7716  ELAINE           GOUDREAUX       CS216          A
301-45-9817  MARVIN           MAYER           CS673          A-
001-45-6677  ROBERT           MACLEAN         EN203          A-
333-45-9900  MIKE             GRANT           MG101          A
301-45-9817  MARVIN           MAYER           MG101          B+

11 records listed.
```

Example 4-28. Sorting a Multivalued Field

▶ BY.EXP.DSND

Description: Sorts on descending multivalued fields.

This keyword works the same as BY.EXP except that values are sorted in descending (or reverse) order. Like BY.EXP, it is intended for use with multivalued fields only. Example 4-29 shows how it works.

```
:LIST STUDENTS BY.EXP.DSND COURSES

LIST STUDENTS BY.EXP.DSND COURSES 17:15:18  09-24-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

333-45-9900  MIKE             GRANT           MG101          A
301-45-9817  MARVIN           MAYER           MG101          B+
001-45-6677  ROBERT           MACLEAN         EN203          A-
301-45-9817  MARVIN           MAYER           CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS216          A
412-05-7716  ELAINE           GOUDREAUX       CS108          B+
412-05-7716  ELAINE           GOUDREAUX       CS105          B
001-45-6677  ROBERT           MACLEAN         CS101          A
333-45-9900  MIKE             GRANT           CS101          B
301-45-9817  MARVIN           MAYER           CS101          B
141-05-9988  SUSAN            KATZ            CS101          A

11 records listed.
```

Example 4-29. Reverse Sorting on a Multivalued Field

OUTPUT KEYWORDS

The output keywords allow you to override display defaults like column headings, inter-column spacing, automatic record ID display, and so forth. These keywords are intended for use with the LIST and SORT verbs.

▶ COL-SUPP

Description: Suppresses default column headings.

Synonyms: COL.SUP

In each default display, INFORM puts the display names associated with each field to be displayed over the column containing the field values for that descriptor. Examples 4-30 and 4-31 show displays with and without column headings.

```
:LIST STUDENTS WITH GPA GT 3.3

LIST STUDENTS WITH GPA GT 3.3  14:04:00  09-06-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              MG101          B+
                                              CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                              CS108          B+
                                              CS216          A
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-
333-45-9900  MIKE             GRANT           CS101          B
                                              MG101          A

5 records listed.
```

Example 4-30. Display With Default Column Headings

```
:LIST STUDENTS LNAME WITH GPA GT 3.3 COL-SUPP

LIST STUDENTS WITH GPA GT 3.3 COL-SUPP 14:03:51  09-06-81  PAGE     1

301-45-9817   MARVIN          MAYER          CS101      B
                                             MG101      B+
                                             CS673      A-
412-05-7716   ELAINE          GOUDREAUX      CS105      B
                                             CS108      B+
                                             CS216      A
141-05-9988   SUSAN           KATZ           CS101      A
001-45-6677   ROBERT          MACLEAN        CS101      A
                                             EN203      A-
333-45-9900   MIKE            GRANT          CS101      B
                                             MG101      A

5 records listed.
```

Example 4-31. Display With Column Heading Suppression

▶ COL.SUP

Description: Suppresses default column headings.

Synonyms: COL-SUPP

See COL-SUPP above.

▶ DBL-SPC

Description: Double spaces output records.

Synonyms: DBL.SPC

This keyword overrides the default single-spacing between output records. Example 4-32 shows a display with single-spacing between records, while Example 4-33 shows a report with double-spacing between the same records. The GT.B descriptor used in these examples is an I-descriptor that looks for students whose numeric grades (NGRADES) are 3.0 or better.

```
:LIST STUDENTS LNAME FNAME NGRADES GT.B WITH GT.B EQ 2

LIST STUDENTS LNAME FNAME NGRADES GT.B WITH GT.B EQ 2 14:04:57  09-06-81  PAGE
     1
SOC-SEC-NO. LAST NAME..... FIRST NAME..... NUMERIC GRADE GRADES ABOVE B

301-45-9817 MAYER          MARVIN          3.0           2
                                           3.3
                                           3.7
412-05-7716 GOUDREAUX      ELAINE          3.0           2
                                           3.3
                                           4.0
001-45-6677 MACLEAN        ROBERT          4.0           2
                                           3.7

3 records listed.
```

Example 4-32. Default Inter-record Spacing

```
:LIST STUDENTS LNAME FNAME NGRADES GT.B WITH GT.B EQ 2 DBL-SPC

LIST STUDENTS LNAME FNAME NGRADES GT.B WITH GT.B EQ 2 DBL-SPC
14:05:44  09-06-81  PAGE     1
SOC-SEC-NO. LAST NAME..... FIRST NAME..... NUMERIC GRADE GRADES ABOVE B

301-45-9817 MAYER          MARVIN          3.0           2
                                           3.3
                                           3.7

412-05-7716 GOUDREAUX      ELAINE          3.0           2
                                           3.3
                                           4.0

001-45-6677 MACLEAN        ROBERT          4.0           2
                                           3.7


3 records listed.
```

Example 4-33. Double-spacing Between Records

▶ DBL.SPC

Description: Double spaces output records.

Synonyms: DBL-SPC

See DBL-SPC above.

▶ HDR-SUPP

Description: Suppresses default heading.

Synonym: HDR.SUP

Whenever an INFORM sentence produces a display of one or more records from a file, the output is introduced by a default header. This header contains the text of the executed INFORM sentence, the time and date, and the page number of the display. The HDR-SUPP keyword suppresses the header, as shown in Example 4-34.

```
:LIST STUDENTS WITH GPA GT 3.0 AND COURSES LIKE CS...

LIST STUDENTS WITH GPA GT 3.0 AND COURSES LIKE CS... 14:07:06  09-06-81  PAGE
   1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              MG101          B+
                                              CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                              CS108          B+
                                              CS216          A
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-
333-45-9900  MIKE             GRANT           CS101          B
                                              MG101          A

5 records listed.
```

Example 4-34. Default INFORM Headers

Example 4-35 illustrates what happens when the HDR.SUP is used.

```
:LIST STUDENTS WITH GPA GT 3.0 AND COURSES LIKE CS... HDR.SUP

SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                             MG101          B+
                                             CS673          A-
412-05-7716  ELAINE           GOUDREAUX       CS105          B
                                             CS108          B+
                                             CS216          A
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
                                             EN203          A-
333-45-9900  MIKE             GRANT           CS101          B
                                             MG101          A

5 records listed.
```

Example 4-35. Display With Header Suppression

▶ HDR.SUP

Description: Suppresses default heading.

Synonyms: HDR-SUPP

Same as HDR-SUPP above.

▶ ID-SUP

Description: Suppresses record ID display.

Synonyms: ID.SUP

In all record output displays, INFORM automatically includes the record ID values of any records displayed. Record ID values are always displayed in the first column of the output. To suppress this display, use the ID-SUP or ID.SUP keywords. This will cancel the automatic display of record ID values, as shown in Example 4-36.

```
:LIST STUDENTS WITH LNAME AFTER K AND GPA GT 3.0

LIST STUDENTS WITH LNAME AFTER K AND GPA GT 3.0 14:10:12  09-06-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

301-45-9817  MARVIN           MAYER           CS101          B
                                              MG101          B+
                                              CS673          A-
141-05-9988  SUSAN            KATZ            CS101          A
001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-

3 records listed.
```

Example 4-36. Default Record ID Display

Example 4-37 shows what the screen shown in Example 4-36 would look like with ID suppresion:

```
:LIST STUDENTS WITH LNAME AFTER K AND GPA GT 3.0 ID.SUP

LIST STUDENTS WITH LNAME AFTER K AND GPA GT 3.0 ID.SUP 14:10:19  09-06-81  PAGE
    1
FIRST NAME.....   LAST NAME.....   COURSE NUMBER   COURSE GRADE

MARVIN            MAYER            CS101           B
                                   MG101           B+
                                   CS673           A-
SUSAN             KATZ             CS101           A
ROBERT            MACLEAN          CS101           A
                                   EN203           A-

3 records listed.
```

Example 4-37. Display With ID.SUP

▶ ID.SUP

Description: Suppresses record ID display.

Synonyms: ID-SUP

See ID-SUP above.

Note

The ID.SUP keyword is used in the @ phrase of the sample COSTS file to suppress the COURSE.NO field in default displays.

▶ LPTR

Description: Prints output on line printer.

When appended to a LIST, SORT, or SUM sentence, the LPTR keyword sends the output that would normally go to the terminal screen to the line printer instead.

The LPTR keyword is used in the following manner:

verb filename[options]LPTR [nn]

The verb can be LIST, SORT, SELECT, SSELECT or SUM, and options can be any keywords, field names, and so forth included in a typical INFORM sentence. The LPTR keyword can appear anywhere in the sentence, but is most often tacked onto the end, after the sentence has been perfected and a hard copy of the resulting output is desired. If there is more than one printer on your system, use the nn option to specify at which logical printer you want the output to be printed. Decimal numbers from 0 to 255 are supported. The default logical printer number is 0.

▶ NO.PAGE

Description: Suppresses page pause.

Synonyms: NOPAGE

Normally, when INFORM displays a set of information that is too lengthy to fit on a single terminal screen, it breaks up the output into segments of 23 lines each. After each set of 23 lines, which is called one logical screen page, has been displayed, INFORM prompts you with the following message:

Press <NEW LINE> to continue...

INFORM then waits for you to hit RETURN or NEWLINE before displaying the next logical page. If you hit a Q in response to this prompt, the next page will not be displayed. The NO.PAGE keyword tells INFORM to display the entire set of output information without pausing for logical page breaks. This makes the output "scroll" past in an uninterrupted stream.

It is difficult to represent this on hard copy, so try it yourself to see the results.

▶ NOPAGE

Description: Suppresses page pause.

Synonyms: NO.PAGE

See NO.PAGE above.

▶ VERT

Description: Displays output in vertical format.

Synonyms: VERTICALLY

The usual output format is columnar, with the data neatly lined up under column headings (which are merely the display names for these fields) in columns across the terminal screen. If the data is too wide to fit across the screen, INFORM automatically displays the output in vertical format. In the vertical format, all the information for a single record is displayed together, with no formatting or spacing between lines of field data.

Multivalued field data is listed out in columns underneath the display names. If multivalued fields are linked in an association, the values for these fields are displayed side-by-side in columns.

To force this form of output even if the display could fit in the default column format, use the VERT or VERTICALLY keywords. Examples 4-38 and 4-49 show the difference between the default columnar format and the vertical format. In the first example, a record is displayed using the default column type format. The second example, 4-39, shows the same record displayed with the VERT keyword.

```
:LIST STUDENTS WITH GT.B EQ 3

LIST STUDENTS WITH GT.B EQ 3 14:10:50  09-06-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

131-43-5670  ADELLE           JARVIS          CS101          B
                                             CS301          A
                                             CS579          A-
                                             CS673          D
                                             CS673          B+

One record listed.
```

Example 4-38. Default Column Display

```
:LIST STUDENTS WITH GT.B EQ 3 VERT

LIST STUDENTS WITH GT.B EQ 3 VERT 14:11:00  09-06-81  PAGE     1
SOC-SEC-NO. 131-43-5670
FIRST NAME.. ADELLE
LAST NAME.. JARVIS
COURSE NUMBER COURSE GRADE
CS101         B
CS301         A
CS579         A-
CS673         D
CS673         B+

One record listed.
```

Example 4-39. Vertical Display Format

▶ VERTICALLY

Description: Displays output in vertical format.

Synonyms: VERT

See VERT above.

REPORT KEYWORDS

Creating a report in INFORMATION is quite easy to do. There is no need for tedious measurements or programming. Experimentation is very simple because every time you make a change you don't have to compile, reload, and re-execute a program. You can save your report specifications in a sentence or paragraph in your VOC file. That way it is easy to produce a report, printed and ready, by simply typing the name of the stored sentence or paragraph. See Storing INFORM Sentences in Section 1 for details.

INFORM does so much default formatting for you that very little adjustment is necessary to achieve the results you have in mind. With the output keywords which have just been described, and the various report keywords, you can do nearly any type of output formatting. The output keywords allow you to override INFORM's defaults and the report keywords let you replace the default values with your own.

▶ AVERAGE

Description: Calculates average for numeric fields.

Synonyms: AVG

The AVERAGE keyword performs simple arithmetic on numeric field values. Within a given set of records, which may represent all the records in the file or those that match a certain stated condition, you can obtain the average value over the set for a particular numeric field by using the AVERAGE keyword.

For example, in the STUDENTS file, a field called TUITION was defined. This field is an I-descriptor that calculates student tuition charges by adding up all the course costs in each record. To find the average student's tuition bill, use the sentence shown in Example 4-40.

```
:LIST STUDENTS AVERAGE TUITION

LIST STUDENTS AVERAGE TUITION 14:11:27  09-06-81  PAGE     1
SOC-SEC-NO.    TUITION COSTS

301-45-9817        $1,050.00
412-05-7716          $675.00
131-43-5670        $2,025.00
111-45-5566          $300.00
343-05-9988          $650.00
141-05-9988          $300.00
001-45-6677          $300.00
333-45-9900          $575.00
                   ==========
                     $734.00

8 records listed.
```

Example 4-40. Obtaining Averages

▶ AVG

Description: Calculates average for field.

Synonym: AVERAGE

See AVERAGE above.

▶ BREAK-ON

Description: Specifies breakpoint in report.

Synonym: BREAK.ON

The breakpoint specifiers, BREAK.ON and BREAK.SUP, (described below) and their respective synonyms, make it possible to highlight a certain field in a report, and to obtain subtotals, partial averages, and partial percentages for numeric fields whenever the value of the breakpoint field changes. Every time a breakpoint field changes value, it is highlighted by a row of stars (***) underneath the last occurrence of that value. The next value for that field will be different from the previous one. This can be changed, using the BREAK.ON options which are discussed below. At each breakpoint change, INFORM skips a line between the output from one record and the output for the next record, unless you tell it to do otherwise.

Normally, the field being breakpointed is also sorted, so that all identical values for a particular field will be processed and displayed together. The changes in values is much easier to process visually if the output is sorted on the breakpoint field. Breakpoints are generally used in conjunction with the TOTAL, CALCULATE, AVERAGE, or PCT keywords. If these keywords are included in a sentence with a breakpoint specifier, then the appropriate intermediate TOTALs, AVERAGEs, PERCENTs, will be displayed on the breakpoint line.

Breakpoints and Intermediate Values

Breakpoints are used with the TOTAL, AVERAGE, CALCULATE, and PERCENT keywords so that intermediate totals, averages, and percents can be displayed whenever the value of the designated breakpoint field changes. Breakpointing a report on such a field allows you to obtain subtotal or intermediate information on breakpoint lines, as well as grand totals, final averages, or overall percent information at the bottom of the report. See CALC and TOTAL for more information on breakpoints and calculations.

Specifying Breakpoints

A breakpoint expression, which consists of BREAK.ON or BREAK-ON and an optional breakpoint clause, can appear in any LIST or SORT sentence. The breakpoint clause specifies the options you want to use. A breakpoint expression has the following general format:

```
{BREAK.ON}{["['opt[opt]...']  [text]"] field.name
{BREAK-ON}
```

The options, indicated by opt, should be packed together within single quotes, as in 'BDL'. The entire breakpoint clause is optional, but when specified, it can include text only, options only, or a combination of both. The ellipsis (...) indicates that you can put more than one option within a pair of matching single quotes. Any text you want printed out in place of the row of stars can be placed anywhere in the options clause.

Rules for the Options Clause

When using the options clause, follow these rules:

- Enclose the entire breakpoint clause in double quotes.
- Enclose all options in single quotes (e.g., 'BD').
- Put the breakpoint field.name immediately after the breakpoint option expression.

The breakpoint options are described below.

Breakpoint Options

The BREAK.ON and BREAK-ON keywords take the following options:

Option	Function
B	Causes current breakpoint value to be used in the HEADING or FOOTING line where indicated by the HEADING or FOOTING "B" option. A new page is generated every time the breakpoint value changes. Only the first BREAK.ON "B" field in a sentence will be used. See FOOTING and HEADING keywords below.
D	Suppresses printing of the breakpoint line if there is only one line of detail. If used with multivalued fields, it works best with BY.EXP[DSND] sort key so breakpointing is done properly. This option causes no blank lines to occur between output records.
L	Eliminates display of breakpoint line entirely, but skips are still generated between value changes. If specified, text is ignored.
P	Causes a new page to be started for each new breakpoint value.
V	Inserts value of the breakpoint field in place of the row of stars that are normally printed there.

Inclusion of any one of the above options in a BREAK.ON or BREAK-ON clause causes the the "stars and bars" line to be suppressed in the output. The stars are printed out beneath the breakpoint field value and the bars are appear beneath any fields being averaged, totalled or percentaged. The appropriate intermediate values for these calculations appear beneath the bars. A single line of bars is printed above each intermediate total, average, or percentage, while a double line of bars is printed over grand totals, final averages, and final percentages.

Breakpoint Examples

Here is an example of how breakpointing might be used in a typical report. The file used contains inventory information for a clothing store. The @ID field contains the item number, the ITEM field describes the garment, the LAND.COST field lists the initial cost of the item, the PRICE field represents the actual retail price of the article, and the FABRIC field tells in what fabric the item is available. The MARKUP field is an I-descriptor which calculates the percent markup represented by the retail price of each item. MARKUP is defined as:

```
TOTAL(LAND.COST)/TOTAL(PRICE)*100
```

The report shown in Example 4-41 groups the different clothing items according to fabric so that the markup patterns relating to fabric type can be noted. Note that the breakpoint field in this report is FABRIC, and that the output is sorted on this field as well.

```
:LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL LAND.COST TOTAL PRICE

LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL LAND.COST TOTAL PRICE
 18:42:27  10-09-81  PAGE     1
GARMENTS    FABRIC.....     LANDED COST    PRICE...

46ELS       COTTON                $30.00     $95.00
90CPT       COTTON                $65.00    $110.00
54CDR       COTTON               $150.00    $300.00
            **                 ----------  ----------
            COTTON               $245.00    $505.00

91CPT       SILK                  $90.00    $170.00
54DR        SILK                 $200.00    $400.00
            **                 ----------  ----------
            SILK                 $290.00    $570.00

44ELS       WOOL                  $45.00    $110.00
            **                 ----------  ----------
            WOOL                  $45.00    $110.00

42ELS       WOOL CREPE            $45.00    $125.00
            **                 ----------  ----------
            WOOL CREPE            $45.00    $125.00

Press <NEW LINE> to continue...
LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL LAND.COST TOTAL PRICE
 18:42:27  10-09-81  PAGE     2
GARMENTS    FABRIC.....     LANDED COST    PRICE...

                            ==========  ==========
                              $625.00    $1310.00


7 records listed.
```

Example 4-41. A Report With Breakpointing

BREAK.ON With Text: The next report, Example 4-42, shows the use of BREAK.ON with text, which is inserted where the row of stars would normally appear.

```
:LIST GARMENTS BY FABRIC BREAK.SUP FABRIC TOTAL LAND.COST TOTAL PRICE

LIST GARMENTS BY FABRIC BREAK.SUP FABRIC TOTAL LAND.COST TOTAL PRICE
 10:36:32  10-12-81  PAGE     1
GARMENTS     LANDED COST     PRICE...

46BLS             $30.00      $95.00
90CPT             $65.00     $110.00
54CDR            $150.00     $300.00
                 --------    --------
                 $245.00     $505.00

91CPT             $90.00     $170.00
54DR             $200.00     $400.00
                 --------    --------
                 $290.00     $570.00

44BLS             $45.00     $110.00
                 --------    --------
                  $45.00     $110.00

42BLS             $45.00     $125.00
                 --------    --------
                  $45.00     $125.00

Press <NEW LINE> to continue...
LIST GARMENTS BY FABRIC BREAK.SUP FABRIC TOTAL LAND.COST TOTAL PRICE
 10:36:32  10-12-81  PAGE     2
GARMENTS     LANDED COST     PRICE...

                 ========    ========
                 $625.00    $1310.00

7 records listed.
```

Example 4-42. BREAK.ON With Text Option

More Breakpoint Options: Example 4-43 shows the use of the P option, which causes a new page to be generated for every change in breakpoint value, and the V option which puts the value of the breakpoint field where the row of stars is ordinarily printed. Although a separate screen page is generated for each record displayed, only the first two pages of the display are shown in this example. Note that the breakpoint field, FABRIC, is sorted so the report groups all the silks, cottons, and wools together.

```
:LIST GARMENTS BY FABRIC BREAK.ON "'PV'" FABRIC TOTAL PRICE

LIST GARMENTS BY FABRIC BREAK.ON ""PV"" FABRIC TOTAL PRICE 18:44:26  10-09-81
PAGE     1
GARMENTS    FABRIC.....    PRICE...

46BLS       COTTON           $95.00
90CPT       COTTON          $110.00
54CDR       COTTON          $300.00
            COTTON          -------
            COTTON          $505.00
Press <NEW LINE> to continue...(CR)
LIST GARMENTS BY FABRIC BREAK.ON ""PV"" FABRIC TOTAL PRICE 18:44:26  10-09-81
PAGE     2
GARMENTS    FABRIC.....    PRICE...

91CPT       SILK            $170.00
54DR        SILK            $400.00
            SILK            -------
            SILK            $570.00
Press <NEW LINE> to continue...(CR)
LIST GARMENTS BY FABRIC BREAK.ON ""PV"" FABRIC TOTAL PRICE 18:44:26  10-09-81
PAGE     3
GARMENTS    FABRIC.....    PRICE...

44BLS       WOOL            $110.00
            WOOL            -------
            WOOL            $110.00
Press <NEW LINE> to continue...Q
```

Example 4-43. Some Breakpoint Options

BREAK.ON and Multivalued Fields: Multivalued fields which are used as breakpoint fields should be sorted using the BY.EXP or BY.EXP.DNSD keywords, as shown in Example 4-44. The file used in this example contains similar information to the GARMENTS file which is used in previous examples. The information is just organized somewhat differently. FABRIC, for example, is a multivalued field, so that the same garment can appear in more than one fabric.

```
:LIST INVENTORY BY.EXP FABRIC BREAK.ON FABRIC DESC QTY TOTAL PRICE

LIST INVENTORY BY.EXP FABRIC BREAK.ON FABRIC DESC QTY TOTAL PRICE
 18:45:16  10-09-81  PAGE      1
INVENTORY..... FABRIC..... ITEM DESC..... QTY.  PRICE...

502DRS         COTTON      2-PC SLIT          5  $300.00
602PTS         COTTON      BUTTON-CUFF PANTS 10  $120.00
               **                                ---------
               COTTON                            $420.00

602PTS         LT. WOOL    BUTTON-CUFF PANTS  8  $225.00
               **                                ---------
               LT. WOOL                          $225.00

602PTS         RAW SILK    BUTTON-CUFF PANTS  8  $210.00
               **                                ---------
               RAW SILK                          $210.00

502DRS         SILK        2-PC SLIT          4  $400.00
               **                                ---------
               SILK                              $400.00

                                                 =========
                                                 $1255.00
```

Example 4-44. Breakpointing a Multivalued Field

BREAK.ON With D Option: The BREAK.ON D option suppresses the display of the breakpoint line in a report if there is only one line of detail. This means that if there is only one occurrence of a particular breakpoint field value, the breakpoint line, including the row of stars and any subtotals, partial averages, or partial percentages that might be relevant, will be suppressed. The sentence executed in the above example is changed by adding the D option to the BREAK.ON clause, resulting in the display shown in Example 4-45.

```
:LIST GARMENTS BY FABRIC BREAK.ON "'D'" FABRIC TOTAL PRICE

LIST GARMENTS BY FABRIC BREAK.ON ""D"" FABRIC TOTAL PRICE 18:46:00  10-09-81  P
AGE     1
GARMENTS    FABRIC.....     PRICE...

46BLS       COTTON            $95.00
90CPT       COTTON           $110.00
54CDR       COTTON           $300.00
                             --------
            COTTON           $505.00

91CPT       SILK             $170.00
54DR        SILK             $400.00
                             --------
            SILK             $570.00

44BLS       WOOL             $110.00

42BLS       WOOL CREPE       $125.00

                             ========
                            $1310.00
```

Example 4-45. BREAK.ON With the D Option

For more ideas on how to use breakpoints, see the examples accompanying the CALC and TOTAL keywords.

▶ BREAK.ON

Description: Specifies breakpoints in reports.

Synonyms: BREAK-ON

See BREAK-ON above.

▶ BREAK.SUP

Description: Suppresses breakpoint display.

BREAK.SUP performs the same breakpoint action as BREAK.ON, but it suppresses the actual display of all breakpoint field values and of the row of stars that usually follows each breakpoint change. The output is still arranged according to breakpoint field values, even though the values themselves are not printed out. The BREAK.SUP keyword should be used in this format:

```
LIST filename BY field.name BREAK.SUP ["'opts'"] field.name
```

As in BREAK.ON, the field to be used as a breakpoint field should be sorted. The options (opts) can be any of the following:

Option	Function
B	Causes current breakpoint value to be used in the HEADING or FOOTING line where indicated by the HEADING or FOOTING "B" option. A new page is generated every time the breakpoint value changes. Only the first BREAK.SUP "B" field in a sentence will be used. See the FOOTING and HEADING keywords below.
D	If there is only one line of detail, the breakpoint line will be suppressed entirely. When no subtotals, averages, percents, etc., are being calculated, the breakpoint line is simply a blank line. The D option will suppress the blank line if there is only one line of detail.
P	Causes a new page to be started for each new breakpoint value.

Example 4-46 shows the use of the BREAK.SUP keyword. The sentence is the same as that used in Example 4-41, except that BREAK.ON has been replaced by BREAK.SUP.

```
:LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL PRICE TOTAL LAND.COST

LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL PRICE TOTAL LAND.COST
 12:16:22  10-12-81  PAGE      1
GARMENTS    FABRIC.....     PRICE...     LANDED COST

46BLS       COTTON            $95.00          $30.00
90CPT       COTTON           $110.00          $65.00
54CDR       COTTON           $300.00         $150.00
            **               --------        --------
            COTTON           $505.00         $245.00

54SDR       SILK             $400.00         $200.00
69PTS       SILK             $250.00         $175.00
91CPT       SILK             $170.00          $90.00
            **               --------        --------
            SILK             $820.00         $465.00

44ELS       WOOL CREPE       $110.00          $45.00
42ELS       WOOL CREPE       $125.00          $45.00
            **               --------        --------
            WOOL CREPE       $235.00          $90.00

                             ========        ========
                            $1560.00         $800.00
```

Example 4-46. The BREAK.SUP Keyword

▶ CALC

Description: Specifies TOTAL calculations.

Synonyms: CALCULATE

The CALC and CALCULATE keywords are meant to be used with I-descriptors that contain the TOTAL function. The TOTAL keyword, described below, is different from the TOTAL function, which was described earlier in Section 2.

CALC and the TOTAL Function

The CALC keyword can be used with breakpointing to obtain intermediate values for I-descriptor expressions which contain the TOTAL function. This only makes sense, however, when the I-descriptor expression defines more than one total to be accumulated. These totals are then combined to yield a single value for this I-descriptor expression. This means that the TOTAL function should appear more than once in an I-descriptor expression. If there is only one total being accumulated in the expression, the CALC keyword is not relevant, and will do nothing special. CALC only produces subtotals on breakpoint lines (if breakpoints are specified in the report) when an I-descriptor expression defines two or more independent totals.

How CALC and TOTAL Work: When breakpointing is not in effect, the TOTAL function accumulates running totals on field information and normally prints out the grand total, or final evaluation of the I-descriptor expression at the bottom of the report page. However, when used with breakpointing and the CALC keyword, intermediate values for such an I-descriptor expression can be obtained on each breakpoint line. The individually accumulated subtotals will be used to calculate an intermediate value for the entire expression at each breakpoint change. A final value for the I-descriptor expression will also be printed at the bottom of the report.

Example of TOTAL and CALC: The following examples show the impact of using the CALC keyword with an I-descriptor that accumulates more than one total. Both examples use the MARKUP I-descriptor in the GARMENTS file, which is defined as:

```
TOTAL(LAND.COST)/TOTAL(RETAIL) * 100
```

This expression defines two individual totals that must be accumulated separately.

The following examples, 4-47 and 4-48, show how CALC works. The first part of Example 4-47 shows the results of evaluating this descriptor without using CALC and without breakpoints. The value for MARKUP is evaluated for each item and printed out. The second part of Example 4-47 shows the use of CALC and AVG with the MARKUP field. Notice that the final results are not equivalent because the value for CALC MARKUP is evaluated differently.

```
:LIST GARMENTS BY FABRIC ITEM LAND.COST PRICE MARKUP

LIST GARMENTS BY FABRIC ITEM LAND.COST PRICE MARKUP 11:30:01  10-12-81  PAGE
   1
GARMENTS    ITEM.....          LANDED COST     PRICE...     MARKUP

46BLS       TUNIC                    $30.00       $95.00      32.00
90CPT       CUFFED PANT              $65.00      $110.00      59.00
54CDR       COLLARED DRESS          $150.00      $300.00      50.00
54SDR       SLIT DRESS              $200.00      $400.00      50.00
69PTS       BLOOMERS                $175.00      $250.00      70.00
91CPT       CUFFED PANT              $90.00      $170.00      53.00
44BLS       TUNIC                    $45.00      $110.00      41.00
42BLS       RUFFLE BLOUSE            $45.00      $125.00      36.00

8 records listed.

:LIST GARMENTS BY FABRIC ITEM LAND.COST PRICE CALC MARKUP AVG MARKUP

LIST GARMENTS BY FABRIC ITEM LAND.COST PRICE CALC MARKUP AVG MARKUP
 11:30:16  10-12-81  PAGE     1
GARMENTS    ITEM.....          LANDED COST     PRICE...     MARKUP     AVG MARKUP

46BLS       TUNIC                    $30.00       $95.00      32.00          32.00
90CPT       CUFFED PANT              $65.00      $110.00      59.00          59.00
54CDR       COLLARED DRESS          $150.00      $300.00      50.00          50.00
54SDR       SLIT DRESS              $200.00      $400.00      50.00          50.00
69PTS       BLOOMERS                $175.00      $250.00      70.00          70.00
91CPT       CUFFED PANT              $90.00      $170.00      53.00          53.00
44BLS       TUNIC                    $45.00      $110.00      41.00          41.00
42BLS       RUFFLE BLOUSE            $45.00      $125.00      36.00          36.00
                                                                        ==========
                                                              51.00          49.00

8 records listed.
```

Example 4-47. An I-descriptor With Multiple TOTALs

Example 4-48 shows how CALC works with breakpointing. The value for MARKUP is calculated for each fabric type, that is, silk, wool crepe, and cotton. Thus, four different CALC MARKUP field values are shown: three partial values on breakpoint lines and one overall value, known as the grand total, at the bottom of the report.

```
:LIST GARMENTS BY FABRIC BREAK.ON FABRIC ITEM LAND.COST PRICE CALC MARKUP AVG MARKUP

LIST GARMENTS BY FABRIC BREAK.ON FABRIC ITEM LAND.COST PRICE CALC MARKUP AVG
MARKUP 11:30:52  10-12-81  PAGE     1
GARMENTS FABRIC..... ITEM..... LANDED COST PRICE... MARKUP AVG MARKUP

46BLS   COTTON          TUNIC              $30.00   $95.00  32.00     32.00
90CPT   COTTON          CUFFED PANT        $65.00  $110.00  59.00     59.00
54CDR   COTTON          COLLARED DRESS    $150.00  $300.00  50.00     50.00
        **
        COTTON                                               49.00     47.00

54SDR   SILK            SLIT DRESS        $200.00  $400.00  50.00     50.00
69PTS   SILK            BLOOMERS          $175.00  $250.00  70.00     70.00
91CPT   SILK            CUFFED PANT        $90.00  $170.00  53.00     53.00
        **
        SILK                                                 57.00     58.00

44BLS   WOOL CREPE       TUNIC              $45.00  $110.00  41.00     41.00
42BLS   WOOL CREPE       RUFFLE BLOUSE      $45.00  $125.00  36.00     36.00
        **
        WOOL CREPE                                           38.00     38.00

                                                             51.00     49.00
```

Example 4-48. CALC and the TOTAL Function

Some Remarks on CALC

CALC should only be used with I-descriptors that define more than one total which must be separately accumulated in order to obtain a final expression value.

CALC can be specified with I-descriptors that do not contain the TOTAL function, and with D-type descriptors. In these cases, only the detail information for these descriptors will be printed out and nothing will be printed on the breakpoint line for these fields, if breakpointing is specified in the sentence. The value of the CALC field will always be ignored on the breakpoint line if the field is an I-descriptor that doesn't contain the TOTAL function or is a D-descriptor.

Note

If an INFORM sentence that includes breakpointing references an I-descriptor which accumulates multiple TOTALs, and the CALC keyword is not specified for this field, no accumulated subtotal or partial value information will be printed on the breakpoint line. Therefore, it is necessary to use CALC with such a descriptor if you want intermediate values for this expression to be displayed with each breakpoint value change.

▶ CALCULATE

Description: Specifies TOTAL calculations.

Synonym: CALC

See CALC above.

▶ COL.SPACES

Description: Specifies inter-column spacing.

Synonyms: COL.SPCS

INFORM normally puts from one to four spaces between each column header in the horizontal (default) output display. The number of spaces that appears between the columns depends both on the total width needed for the fields to be displayed, and on the length of the display names for each of the fields. Display names are defined in Field 4 of each descriptor record.

To override this default spacing, use COL.SPACES (or COL.SPCS) with a numeric argument specifying the number of spaces you want between columns. Used without a numeric argument, these keywords will force one space between each column.

Examples 4-49 and 4-50 show two displays, the first using INFORM'S default column spacing, and the second one using the COL.SPACES keyword.

```
:LIST STUDENTS FNAME LNAME COURSES GPA

LIST STUDENTS FNAME LNAME COURSES GPA  15:54:11  09-06-81  PAGE     1
SOC-SEC-NO.   FIRST NAME.....   LAST NAME.....   COURSE NUMBER   GPA..

301-45-9817   MARVIN            MAYER            CS101           3.33
                                                 MG101
                                                 CS673
412-05-7716   ELAINE            GOUDREAUX        CS105           3.43
                                                 CS108
                                                 CS216
131-43-5670   ADELLE            JARVIS           CS101           3.00
                                                 CS301
                                                 CS579
                                                 CS673
                                                 CS673
111-45-5566   STEVEN            MCLEAN           MG101           3.00
343-05-9988   MARTHA            GRANIPF          CS101           0.00
                                                 CS105
141-05-9988   SUSAN             KATZ             CS101           4.00
001-45-6677   ROBERT            MACLEAN          CS101           3.85
                                                 EN203
333-45-9900   MIKE              GRANT            CS101           3.50
                                                 MG101
```

Example 4-49. Default Column Spacing

```
:LIST STUDENTS FNAME LNAME COURSES GPA  COL.SPCS 5

LIST STUDENTS FNAME LNAME COURSES GPA  COL.SPCS 5 15:54:22  09-06-81  PAGE
 1
SOC-SEC-NO.   FIRST NAME.....   LAST NAME.....   COURSE NUMBER   GPA..

301-45-9817   MARVIN            MAYER            CS101           3.33
                                                 MG101
                                                 CS673
412-05-7716   ELAINE            GOUDREAUX        CS105           3.43
                                                 CS108
                                                 CS216
131-43-5670   ADELLE            JARVIS           CS101           3.00
                                                 CS301
                                                 CS579
                                                 CS673
                                                 CS673
111-45-5566   STEVEN            MCLEAN           MG101           3.00
343-05-9988   MARTHA            GRANIPF          CS101           0.00
                                                 CS105
141-05-9988   SUSAN             KATZ             CS101           4.00
001-45-6677   ROBERT            MACLEAN          CS101           3.85
                                                 EN203
333-45-9900   MIKE              GRANT            CS101           3.50
                                                 MG101
```

Example 4-50. Display With COL.SPCS

If the value supplied with COL.SPACES makes the report larger than the width of the screen, the output will be printed in INFORM's vertical output format instead of in the default column format.

▶ COL.SPCS

Description: Specifies inter-column spacing.

Synonyms: COL.SPACES

See COL.SPACES above.

▶ DET-SUPP

Description: Displays breakpoint lines only.

Synonyms: DET.SUPP

When used in a sentence that specifies breakpoints (using the BREAK.ON or BREAK-ON keywords), DET-SUPP causes only breakpoint line information to be displayed in the report. Only the information normally printed on the "stars and bars" line will be displayed. All other information is suppressed, including the record ID display.

Example 4-51 displays only the intermediate totals and averages for the MARKUP field in the GARMENTS file, using the DET.SUP keyword:

```
:LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL MARKUP AVG MARKUP CALC MARKUP DET.SUP

LIST GARMENTS BY FABRIC BREAK.ON FABRIC TOTAL MARKUP AVG MARKUP CALC MARKUP
DET.SUP 11:37:34  10-12-81  PAGE      1
FABRIC.....     MARKUP     AVG MARKUP     MARKUP

COTTON            141.00          47.00      49.00
SILK              173.00          58.00      57.00
WOOL CREPE         77.00          38.00      38.00
                   =====     ==========
                  391.00          49.00      51.00

3 records listed.
```

Example 4-51. Using the DET.SUP Keyword

▶ DET.SUPP

Description: Displays breakpoint lines only.

Synonyms: DET-SUPP

See DET-SUPP above.

▶ FOOTING

Description: Defines footer for report page.

The FOOTING option makes it possible to put one or more lines of text at the bottom of each physical report page. You can include the current date and/or time, a page number, the filename, and a number of other items in the footer. This format representation shows how text values and options, represented by the opt parameter, can be placed in a FOOTING clause of an INFORM sentence:

```
FOOTING "'opt' [text] ['opt'] [text] ['opt']"
```

The options and text can be put in any order and you can use as many options and pieces of text as you like. The possible values for opt are:

Option	Description
B	Inserts current breakpoint field value wherever "B" option appears in the FOOTER clause. Only valid if used with BREAK.ON "B" option. A separate page is generated for each new breakpoint value.
D	Inserts current date at this point in the footer.
F	Inserts name of file at this location in the footer.
L	Inserts a carriage return and line feed at this point in the footer. Useful for spreading out text over more than one line in the footer.
N	Suppresses default page pause in terminal display.
P	Includes the current page number in the footer of each page.
T	Inserts the current time and date at this spot in the footer.

Example 4-52 shows a report that uses the FOOTING D and L options with text:

```
:LIST GARMENTS BY ITEM FOOTING " GARMENT FILE REPORT 'L' AS OF: 'D' " DBL.SPC

LIST GARMENTS BY ITEM FOOTING " GARMENT FILE REPORT "L" AS OF: "D" " DBL.SPC
 11:45:33  10-12-81  PAGE     1
GARMENTS   ITEM.....   FABRIC.....   LANDED COST   PRICE...

69PTS      BLOOMERS    SILK              $175.00     $250.00

54CDR      COLLARED DRESS   COTTON       $150.00     $300.00

90CPT      CUFFED PANT      COTTON        $65.00     $110.00

91CPT      CUFFED PANT      SILK          $90.00     $170.00

42BLS      RUFFLE BLOUSE    WOOL CREPE    $45.00     $125.00

54SDR      SLIT DRESS       SILK         $200.00     $400.00

44BLS      TUNIC            WOOL CREPE    $45.00     $110.00

46BLS      TUNIC            COTTON        $30.00      $95.00

 GARMENT FILE REPORT
 AS OF: 10-12-81
```

Example 4-52. A Report With the FOOTING Option

Spaces can be put between options to improve the visual appeal of a report footer. Also, in specifying the options clause, make sure you put options like D, F, P, and T, in the order you want them to appear in the footer.

Quotes in Footers

Double quotes should surround the entire FOOTING option expression, while each option letter should be enclosed in single quotes (apostrophes). If a text value contains an apostrophe, such as:

```
MARGARET'S GRADE REPORT
```

use two apostrophes in a row to get it to print:

```
FOOTING "MARGARET''S GRADE REPORT"
```

▶ HEADER

Description: Defines heading for report page.

Synonyms: HEADING

INFORM always prints out a default header over each display of records. This header includes the sentence being executed, the time and date the execution took place, and a default page number. To override this default heading, use the HEADER or HEADING keyword. A heading clause has a format identical to the footing clause:

{HEADER / HEADING} ["'opt' text 'opt' text 'opt'"]

The options and text can be put in any order and you can use as many options and pieces of text as you like. The possible values for opt are:

Option	Description
B	Inserts current breakpoint value at this spot in the report header: used with BREAK.ON "B" option. A new page is formed each time the breakpoint value changes.
D	Inserts current date at this spot in the header.
F	Includes name of file at this point in the header.
L	Inserts a carriage return and line feed at this point in the header. This makes multi-line headings possible.
N	Suppresses default page pause in terminal display.
P	Includes the current page number in the header of each page.
T	Inserts the current time and date at this spot in the report header.

Reminders

Remember to put options like D, F, L, P, and T in the order you want them to appear in the heading. For example, if you want the header to be preceded by the date and followed by the page number, define the heading as:

```
HEADER " 'D' Current Sales List 'P'"
```

Double quotes should surround the entire expression, while all option letters should be enclosed in single quotes (apostrophes). If a text value contains an apostrophe, such as:

```
MARGARET'S GRADE REPORT
```

you will have to use two apostrophes in a row to get it to print:

```
HEADER "MARGARET''S GRADE REPORT"
```

HEADER Examples

Example 4-53 uses the D option and the text option in printing out a report for the GARMENTS file:

```
:LIST GARMENTS BY FABRIC HEADER " 'T' REPORT FOR 'F' FILE"

 12:04:52  10-12-81 REPORT FOR GARMENTS FILE
GARMENTS   ITEM.....      FABRIC.....      LANDED COST   PRICE...

46BLS      TUNIC          COTTON                $30.00     $95.00
90CPT      CUFFED PANT    COTTON                $65.00    $110.00
54CDR      COLLARED DRESS COTTON               $150.00    $300.00
54SDR      SLIT DRESS     SILK                 $200.00    $400.00
69PTS      BLOOMERS       SILK                 $175.00    $250.00
91CPT      CUFFED PANT    SILK                  $90.00    $170.00
44BLS      TUNIC          WOOL CREPE            $45.00    $110.00
42BLS      RUFFLE BLOUSE  WOOL CREPE            $45.00    $125.00

8 records listed.
```

Example 4-53. A Report With the HEADING Option

Heading Options and Breakpoints: Example 4-54 shows the use of the BREAK.ON and HEADING "B" options. The current breakpoint values are printed in the spot where the "B" option appears in the HEADING clause. Note that a new page is printed for each new breakpoint value. Only two of them are shown here, but separate pages are generated each time the breakpoint value changes in the file.

```
:LIST GARMENTS BY FABRIC BREAK.ON "'B' " FABRIC ITEM TOTAL LAND.COST TOTAL PRICE
HEADER " ITEM IN 'B'"

 ITEM IN COTTON
GARMENTS   FABRIC.....   ITEM.....        LANDED COST   PRICE...

46BLS      COTTON        TUNIC                $30.00      $95.00
90CPT      COTTON        CUFFED PANT          $65.00     $110.00
54CDR      COTTON        COLLARED DRESS      $150.00     $300.00
                                             -------     -------
           COTTON                            $245.00     $505.00

Press <NEW LINE> to continue...(CR)
 ITEM IN SILK
GARMENTS   FABRIC.....   ITEM.....        LANDED COST   PRICE...

54SDR      SILK          SLIT DRESS          $200.00     $400.00
69PTS      SILK          BLOOMERS            $175.00     $250.00
91CPT      SILK          CUFFED PANT          $90.00     $170.00
                                             -------     -------
           SILK                              $465.00     $820.00
```

Example 4-54. Report With HEADER and BREAK.ON

▶ HEADING

Description: Defines header for report page.

Synonyms: HEADER

See HEADER above.

▶ ID.ONLY

Description: Displays only record IDs.

Synonyms: ONLY

When the default @ or @LPTR phrases are used in producing a display, this keyword suppresses all other fields named in the phrase except the record ID field. Only the record ID values of the output records will be displayed. When this keyword is used in a sentence that specifies one or more field names for display purposes, it will be ignored.

In Example 4-55, when the ID.ONLY is used, the @ phrase, defined as shown, will be ignored, and only the record ID values will be displayed:

```
LIST DICT STUDENTS "@"

LIST DICT STUDENTS "@" 16:17:11  09-06-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

@              PH   FNAME LNAME
                    COURSES.GRADES

One record listed.

:LIST STUDENTS ID.ONLY

LIST STUDENTS ID.ONLY 16:17:17  09-06-81  PAGE     1
SOC-SEC-NO.

301-45-9817
412-05-7716
131-43-5670
111-45-5566
343-05-9988
141-05-9988
001-45-6677
333-45-9900

8 records listed.
```

Example 4-55. Using the ID.ONLY Keyword

▶ MARGIN

Description: Specifies margin size for report page.

To indent a report from the left margin, use the MARGIN keyword, followed by the number of spaces you want to indent. Normally, INFORM produces reports left-justified, with a logical left margin of 0. Example 4-56 shows an output report with a left margin of 10.

```
:LIST STUDENTS FNAME LNAME GRAD.YR ID,SUP HEADING " EXPECTED DEPARTURE DATE" MARGIN 10

          EXPECTED DEPARTURE DATE
          FIRST NAME.....   LAST NAME.....   GRAD YEAR

          MARVIN            MAYER            1985
          ELAINE            GOUDREAUX        1984
          ADELLE            JARVIS           1983
          STEVEN            MCLEAN           1982
          MARTHA            GRANIFF          1983
          SUSAN             KATZ             1985
          ROBERT            MACLEAN          1984
          MIKE              GRANT            1986

          8 records listed.
```

Example 4-56. Report Using the MARGIN Keyword

▶ ONLY

Description: Displays only record IDs.

Synonym: ID.ONLY

See ID.ONLY above.

▶ PCT

Description: Calculates percents.

Synonyms: PERCENT, PERCENTAGE

When specified with the name of a numeric field, the PERCENT and PERCENTAGE keywords calculate and display, for each record in the output, the percentage of this field's total value (using all the values for this field in the listed records) represented by the field value in this record. The chosen field values from each listed record are combined to form the total value to which each individual record value can be compared to produce a percentage.

The format for using the PERCENT keyword is:

LIST filename { PERCENTAGE / PERCENT / PCT } ['n'] field.name

The value for n, which must be quoted, can be any integer from 0 to 5, inclusive. This specifies the number of digits to appear after the decimal point. The default is 2.

field.name is the name of a numeric field for which the percentage is to be calculated.

Percentage Example: In the STUDENTS file, to calculate the percentage of the total tuition cost represented by each student's charges, use the statement shown in Example 4-57.

```
:LIST STUDENTS TOTAL TUITION PCT TUITION

LIST STUDENTS TOTAL TUITION PCT TUITION 12:13:22  10-12-81  PAGE     1
SOC-SEC-NO.    TUITION COSTS     TUITION COSTS

101-45-9817        $1,050.00             17.95
412-05-7716          $675.00             11.54
333-45-9900          $575.00              9.83
131-43-5670        $2,025.00             34.62
111-45-5566          $300.00              5.13
343-05-9988            $0.00
141-05-9988          $575.00              9.83
001-45-6677          $650.00             11.11
                   ==========          ======
                   $5,850.00            100.00

8 records listed.
```

Example 4-57. Calculating Percentages

Note

Percentages are calculated and displayed to two decimal places unless the n option is included in the PERCENT clause, as shown in the above format. Then the percentages will be displayed to n decimal places.

▶ PERCENT

Description: Calculates percents.

Synonyms: PCT, PERCENTAGE

See PCT above.

▶ PERCENTAGE

Description: Calculates percents.

Synonyms: PCT, PERCENT

See PCT above.

▶ TOTAL

Description: Accumulates and displays totals for numeric fields.

The TOTAL keyword totals up all the selected values for a designated numeric field. It is most often used in conjunction with breakpoints, so that subtotals, as well as grand totals, can be obtained in a report.

Note

> The INFORM TOTAL keyword is not the same thing as the TOTAL function which is described earlier in this section under CALC and introduced in Section 2. When used as a keyword, TOTAL should be placed in an INFORM sentence immediately prior to the field for which totals are to be accumulated.

If breakpointing is used, subtotals for all fields specified with TOTAL are printed on the breakpoint line. The grand total for each field specified with the TOTAL keyword is printed at the bottom of the last page of a report underneath two rows of dashes called "TOTAL bars."

TOTAL and Breakpoints

When used with breakpoints, TOTAL causes the subtotals for each set of breakpoint values to be printed on the appropriate breakpoint lines. This makes it possible to obtain individual totals for each group of records which appear in breakpoint groups. For instance, in the GARMENTS file, the subtotal for PRICE field values in records where FABRIC is SILK, will be printed out on the breakpoint line after the last detail line is printed for this group. Sub-total values are displayed underneath a single row of TOTAL bars. If there is only one detail line, the subtotal value will be identical to the detail value for that breakpoint.

Example of Breakpoints and TOTAL Keyword: Example 4-58 displays the subtotals and grand total for all LAND.COST and PRICE field values in the GARMENTS file:

```
:LIST GARMENTS BY FABRIC BREAK.ON "SUB-TOTAL FOR PRICE" FABRIC TOTAL PRICE


LIST GARMENTS BY FABRIC BREAK.ON "SUB-TOTAL FOR PRICE" FABRIC TOTAL PRICE
 18:43:38  10-09-81  PAGE      1
GARMENTS    FABRIC.........    PRICE...

46BLS       COTTON               $95.00
90CPT       COTTON              $110.00
54CDR       COTTON              $300.00
            SUB-TOTAL FOR PRICE--------
            COTTON              $505.00

91CPT       SILK                $170.00
54DR        SILK                $400.00
            SUB-TOTAL FOR PRICE--------
            SILK                $570.00

44BLS       WOOL                $110.00
            SUB-TOTAL FOR PRICE--------
            WOOL                $110.00

42BLS       WOOL CREPE          $125.00
            SUB-TOTAL FOR PRICE--------
            WOOL CREPE          $125.00

Press <NEW LINE> to continue...
LIST GARMENTS BY FABRIC BREAK.ON "SUB-TOTAL FOR PRICE" FABRIC TOTAL PRICE
 18:43:38  10-09-81  PAGE      2
GARMENTS    FABRIC.........    PRICE...

                               ========
                               $1310.00


7 records listed.
```

Example 4-58. TOTAL Keyword and Breakpoints

MISCELLANEOUS KEYWORDS

Miscellaneous keywords have a variety of uses, like processing just the first few records of a file or for processing a sample selection of records.

▶ BLK

Description: Specifies block size for T.DUMP.

The BLK keyword is used with the T.DUMP command which saves INFORMATION file records from disk to tape. The BLK keyword makes it possible to specify the size of the tape record. The default record (block) size is 8192 bytes. This is also the maximum size. T.LOAD will always display the block size of the tape records as files are being restored from tape. That way you always know the block (record) size of the tape you're restoring from.

▶ DICT

Description: Uses dictionary file instead of data file.

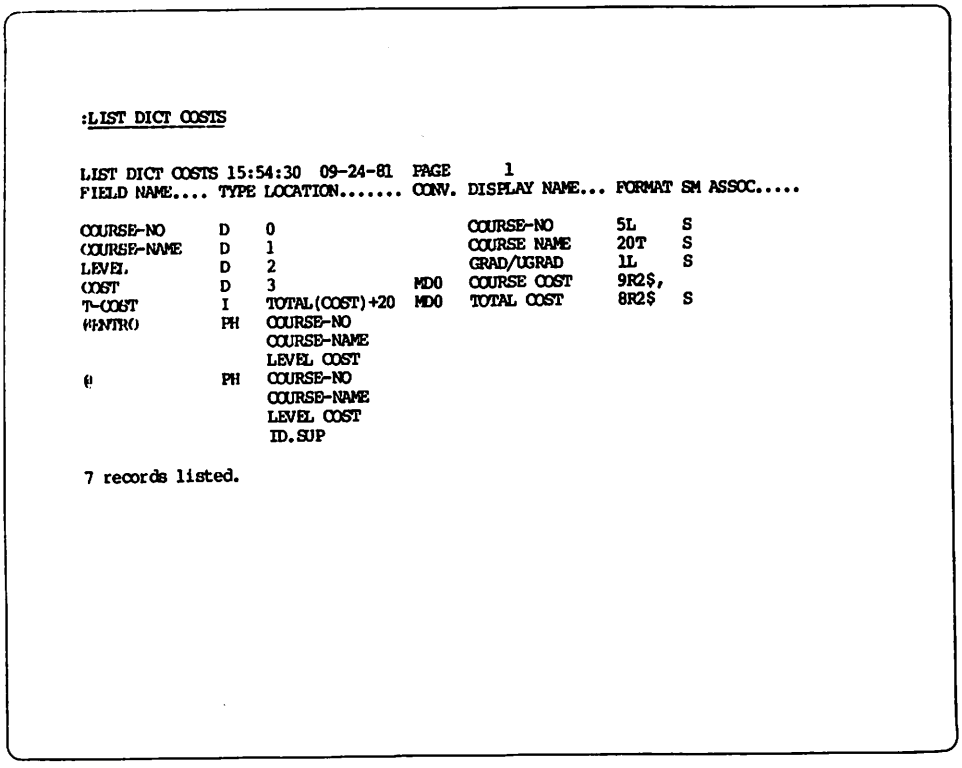
The DICT keyword tells INFORM to use the dictionary file associated with the named file, instead of using the data file.

Example 4-59 shows the records in the COSTS data file:

```
:LIST COSTS

LIST COSTS 15:54:15  09-24-81  PAGE     1
COURSE-NO     COURSE NAME.....     GRAD/UGRAD     COURSE COST

CS101         INTRO TO C.S.        U                 $275.00
MG101         INTRO TO MGMT        U                 $275.00
CS579         DATA BASE CONCEPTS   G                 $475.00
CS673         SOFTWARE ENG         G                 $475.00
CS600         SYSTEMS OPERATION    G                 $475.00
CS301         DATA STRUCTURES      U                 $300.00
CS108         PASCAL PROGRAMMING                     $300.00

7 records listed.
```

Example 4-59. Listing a Data File

The dictionary file associated with this data file is shown in Example 4-60.



```
:LIST DICT COSTS

LIST DICT COSTS 15:54:30  09-24-81  PAGE     1
FIELD NAME.... TYPE LOCATION..... CONV. DISPLAY NAME... FORMAT SM ASSOC.....

COURSE-NO       D   0                    COURSE-NO       5L     S
COURSE-NAME     D   1                    COURSE NAME     20T    S
LEVEL           D   2                    GRAD/UGRAD      1L     S
COST            D   3                MD0 COURSE COST     9R2$,
T-COST          I   TOTAL(COST)+20   MD0 TOTAL COST      8R2$   S
@;INTRO         PH  COURSE-NO
                    COURSE-NAME
                    LEVEL COST
@               PH  COURSE-NO
                    COURSE-NAME
                    LEVEL COST
                    ID.SUP

7 records listed.
```

Example 4-60. Listing the Dictionary File

▶ FIRST

Description: Processes first nn records of the file.

Synonym: SAMPLE

The user specifies a positive integer for nn. The default is 10. If selection criteria are specified, the records meeting the criteria are chosen first and the sample is then done on this set of records.

Example 4-61 shows how a sampling of records can be obtained.

```
:LIST STUDENTS FIRST 3

LIST STUDENTS FIRST 3 12:07:01  08-24-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

141-05-9988  SUSAN            KATZ           CS101          A
301-45-9817  MARVIN           MAYER          CS101          B
                                             MG101          B+
                                             CS673          A-
412-05-7716  ELAINE           GOUDREAUX      CS105          B
                                             CS108          B+
                                             CS216          A

Sample of 3 records listed.
```

Example 4-61. Listing a Sampling of Records

▶ MTU

Description: Specifies tape drive unit, mode, and tracks.

The MTU keyword is used with the T.DUMP and T.LOAD verbs to indicate the number of a tape drive assigned to a user with the T.ATT or ATTACH commands. The format of the MTU keyword is:

MTU [mtu]

where m is the mode (ASCII is assumed), t represents the number of tracks on the tape (7 or 9, where the default is 9), and u is the unit number assigned to this drive. If u is not specified, tape drive 0 is assumed. If the mtu parameter is not supplied, the default values are used.

Unless you have more than one tape drive assigned to you at any given time, it is not necessary to use this keyword, as the default drive number of 0 is always assumed by INFORM.

▶ OVERWRITING

Description: T.LOAD overwrites existing records.

Unless the OVERWRITING keyword is specified in a T.LOAD operation, T.LOAD will check the disk to see if a record already exists before writing it from tape. If the OVERWRITING keyword is used, T.LOAD will simply write everything from tape to disk without first checking to see if the record already exists. Without the OVERWRITING keyword, much time is taken reading the disk before each record can be restored. If you want to overwrite existing records when restoring a tape file to disk, it's always faster to use the OVERWRITING keyword. Even when writing records to a blank file, T.LOAD (without the OVERWRITING keyword) does a search for each record it is about to write prior to actually restoring it. Thus, it saves time to use the OVERWRITING keyword. Obviously, it is desirable to avoid this overhead if you can.

▶ SAMPLE

Description: Processes first nn records from a file.

Synonym: FIRST

See FIRST above.

▶ SAMPLED

Description: Processes every nth record.

The user specifies a positive integer for n. The default is 10. For example, if n is 5, then every fifth record from the file will be selected and processed accordingly. If selection criteria are specified, records meeting the stated conditions are chosen first, and then every nth one of the chosen set will be sampled. SAMPLED is not the same as SAMPLE, which selects a total of nn records, rather than every nth record.

For example, to sample every 5 records from the STUDENTS file when there are 10 records presently in the file, the output might look like that shown in Example 4-62.

```
:LIST STUDENTS SAMPLED 5

LIST STUDENTS SAMPLED 5 12:06:52  08-24-81  PAGE     1
SOC-SEC-NO.  FIRST NAME.....  LAST NAME.....  COURSE NUMBER  COURSE GRADE

001-45-6677  ROBERT           MACLEAN         CS101          A
                                              EN203          A-
123-88-9900  JAMES            COOK

Sample of 2 records listed.
```

Example 4-62. The SAMPLED Keyword